Universität Hamburg

Fachbereich Informatik
Arbeitsbereich Rechnerorganisation

Diplomarbeit
"Entwurf und Realisierung eines parallelen Analysewerkzeuges für Ethernet-basierte Netze"

Betreuer
Prof. Dr. Bernd E. Wolfinger
Dr. Hans-Joachim Mück

Februar 1996

Carsten Benecke
Am Plack 40
21217 Seevetal
Tel. 04105 / 80586

Erklärung:
Ich versichere, daß ich die vorstehende Arbeit selbständig und ohne fremde Hilfe angefertigt habe und keine außer der angegebenen Quellen und Hilfsmittel benutzt habe.

Seevetal, den 08.02.1996

Danksagung:
Ich möchte mich an dieser Stelle bei meinem Erstbetreuer Herrn Prof. Dr. Bernd E. Wolfinger und meinem Zweitbetreuer Dr. Hans-Joachim Mück für ihre hervorragende Betreuung bedanken. Die vielen Denkanstöße und die konstruktive Kritik waren wertvolle Hinweise für die Erstellung dieser Diplomarbeit.

Kurzfassung

In dieser Arbeit wird dargestellt, wie der Einsatz von Parallelverarbeitung bei der Kommunikationsmeßtechnik die Bewältigung zukünftiger Meßaufgaben in Hochgeschwindigkeitsnetzen ermöglicht. Dazu wird ein Konzept der parallelen Protokollverarbeitung an die Anforderungen der Kommunikationsmeßtechnik angepaßt, um auf dieser Grundlage einen Prototypen eines parallelen Meßwerkzeuges zu realisieren.

Inhaltsverzeichnis

1 Analyse von Kommunikationssystemen **10**

1.1 Notwendigkeit für die Analyse von Kommunikationssystemen 10
1.2 Analysemethoden 12
1.3 Messungen am Kommunikationssystem 13
1.3.1 Die Notwendigkeit für Messungen am Kommunikationssystem . 13
1.3.2 Werkzeuge zum Erfassen von Meßdaten 16
1.4 Klassifikation von Monitoren 19
1.4.1 Klassifikation nach der Meßaufgabe 19
1.4.2 Klassifikation nach der Architektur 20
1.4.3 Klassifikation nach dem Funktionsumfang 23
1.4.4 Klassifikation nach dem Meßverfahren 24
1.4.5 Klassifikation nach der Ergebnispräsentation 24
1.4.6 Zusammenfassung der Klassifikation 25

2 Anforderungen an ein Meßwerkzeug **26**

2.1 Anforderungen seitens der Meßaufgabe (Datenerfassung und Analyse) 26
2.1.1 Schicht 1, physikalische Schicht 26
2.1.2 Schicht 2, Datensicherungsschicht 28
2.1.3 Schicht 3, Vermittlungsschicht 29
2.1.4 Schicht 4, Transportschicht 29
2.1.5 Schicht 5–7, Kommunikationssteuerungs-, Darstellungs- und Anwendungsschicht 29
2.1.6 Alternative Schichteneinteilung 30
2.1.7 Schichtunabhängige Anforderungen 30
2.2 Anforderungen an das Filtern der Daten 30
2.3 Anforderungen an das Bewerten der Daten 31
2.4 Anforderungen an das Anzeigen der Daten 31
2.5 Anforderungen an das Verwalten der Daten 32
2.6 Anforderungen der Benutzer 33
2.7 Anforderungen an ein paralleles Meßwerkzeug für Hochgeschwindigkeitsnetze 33
2.7.1 Anforderungen an die Erweiterbarkeit 34
2.7.2 Anforderungen an die Leistungsfähigkeit 34

2.7.3 Anforderungen an die Skalierbarkeit 35
2.7.4 Zusammenfassung der Anforderungen 35

3 Abbilden der Prozesse auf die Hardware 37
3.1 Zuordnungsproblem 37
3.1.1 Zuordnung von Algorithmen zu Prozessoren 38
3.1.2 Formale Präzisierung der Zuordnung 38
3.1.3 Ziele bei der Zuordnung 41
3.1.4 Zeitpunkt der Zuordnung 42
3.2 Verfügbare Hardware und Entwicklungsumgebung 44
3.2.1 Transputer und 'Transputer-Boards' für PCs 44
3.2.2 Entwicklungsumgebung 53
3.3 Parallele Protokollverarbeitung 55
3.3.1 Bewertungskriterien 55
3.3.2 Vergleich zwischen Endsystem und Monitor 56
3.3.3 Klassifikation der parallelen Protokollverarbeitung 57
3.3.4 Bewertung der Konzepte 61

4 Entwurf des Meßwerkzeuges 67
4.1 Restriktionen durch die Hardware 67
4.2 Wahl eines Konzeptes zur parallelen Protokollanalyse 68
4.3 Grobgliederung des Meßwerkzeuges durch Prozesse 69
4.3.1 Die Anzahl der Prozessoren 69
4.3.2 Die wesentlichen Prozesse des Meßwerkzeuges 69
4.3.3 Entwurf der Grobstruktur des Meßwerkzeuges 72
4.4 Entwurf der essentiellen Funktionen 73
4.4.1 Entwurf der essentiellen Funktionen für die Rohdatenerfassung 73
4.4.2 Entwurf der essentiellen Funktionen für die Analyse 74
4.4.3 Entwurf der essentiellen Funktionen für die globale Statistik . . 79
4.4.4 Entwurf der essentiellen Funktionen für den 'Hostrechner' . . . 79
4.4.5 Entwurf essentieller Funktionen zum Filtern 82
4.5 Entwurf der unterstützenden Funktionen 84
4.5.1 Entwurf von Vermittlungsfunktionen und Verteilungsfunktionen 84
4.5.2 Entwurf von Verbindungsmanagement-Funktionen 87
4.5.3 Entwurf der Kommunikationsunterstützung für die 'Hostrechner'-Transputer-Kommunikation 88
4.6 Sonstige unterstützende Funktionen 90
4.6.1 Funktionen zur Ringpuffer-Verwaltung 90
4.6.2 Funktionen zur Symboltabellen-Verwaltung 90
4.6.3 Funktionen zum Umwandeln von Daten 91
4.7 Klassifikation der Funktionen 91

5 Realisierung des Meßwerkzeuges 93
5.1 Zusammenstellen der Funktionen zu Prozessen . . . 93
5.1.1 Prozesse und 'Threads' auf Transputern . . . 94
5.1.2 Darstellen der Prozesse mit Datenflußdiagrammen . . . 94
5.1.3 Die Prozesse des Transputernetzes als Datenflußdiagramm . . . 96
5.2 Verfeinerung der Prozesse . . . 97
5.2.1 Ein allgemeines 'Prozeß-Template' . . . 97
5.2.2 Der Prozeß zur Rohdatenerfassung . . . 98
5.2.3 Der Prozeß 'PDEV' . . . 98
5.2.4 Der Prozeß 'PPAKET' . . . 101
5.2.5 Der Prozeß 'PSTAT' . . . 102
5.2.6 Der 'Hostrechner'-Prozeß . . . 104
5.3 Realisierung der essentiellen Funktionen . . . 105
5.3.1 Realisierung des Gerätetreibers für die Ethernet-Schnittstelle . 105
5.3.2 Realisierung der Analysefunktionen . . . 107
5.3.3 Realisierung der globale Statistikfunktionen . . . 121
5.3.4 Realisierung der Funktionen für den 'Hostrechner' . . . 121
5.3.5 Realisierung der Filterfunktionen . . . 130
5.4 Realisierung der unterstützenden Funktionen . . . 132
5.4.1 Realisierung der Vermittlungsfunktionen . . . 132
5.4.2 Realisierung der Verteilungsfunktionen . . . 134
5.4.3 Anmerkung zu Verbindungsmanagement-Funktionen . . . 135
5.4.4 Realisierung der Kommunikationsunterstützung für den 'Hostrechner' . . . 135
5.5 Anmerkungen zu weiteren Funktionen . . . 137
5.5.1 Realisierung der Ringpuffer-Verwaltung . . . 137
5.5.2 Realisierung der Symboltabellen-Verwaltung . . . 138
5.6 Vorgehen bei Erweiterungen . . . 138
5.6.1 Erweitern des Prototypen um neue Transputer-Prozesse . . . 138
5.6.2 Erweitern des Werkzeuges um die Prozeßklasse 'PNOTHING' . 139
5.7 Anmerkungen zur Realisierung und zu Erweiterungen . . . 140
5.7.1 Verwaltung der Programmquellen mit einem Versionskontrollsystem . . . 140
5.7.2 Wichtige nicht realisierte Funktionen . . . 143
5.7.3 Erweiterungsvorschläge . . . 144
5.7.4 Vergleich mit anderen Werkzeugen der Kommunikationsmeßtechnik . . . 144

6 Arbeiten mit dem Prototypen 146
6.1 Die Benutzerschnittstelle des Prototypen . . . 146
6.1.1 Das Fenster 'Standardausgabe' . . . 147
6.1.2 Das Fenster 'Standardeingabe' . . . 151
6.1.3 Das Fenster 'MBits/s' . . . 151
6.1.4 Das Fenster 'ARP/RARP-Tabelle' . . . 152

6.1.5 Fehler beim Programmstart 152
6.2 Vorgehensweise zur Durchführung eines Meßexperiments 152
6.2.1 Benennen der Messung 153
6.2.2 Definieren eines Filters 153
6.2.3 Auswählen der zu protokollierenden Informationen 153
6.2.4 Durchführen der Messung 153
6.2.5 Beenden einer Messung 153
6.3 Fallbeispiel I: Analysieren von ARP/RARP-Dateneinheiten 154
6.3.1 Name der Messung 155
6.3.2 Definieren eines Filters 156
6.3.3 Auswählen der zu protokollierenden Informationen 156
6.3.4 Starten einer Messung 157
6.3.5 Unterbrechen einer Messung 157
6.3.6 Beenden einer Messung 158
6.4 Fallbeispiel II: Durchführen einer Lastmessung 159
6.4.1 Auswählen der anzuzeigenden Protokolle 159
6.4.2 Das Fenster 'MBit/s' 159

7 Leistungsbewertung des Prototypen 161
7.1 Engpässe des Prototypen 161
7.1.1 Der primäre Engpaß des Prototypen 162
7.1.2 Analyse des primären Engpasses 162
7.1.3 Vorbeugende Maßnahmen 164
7.1.4 Der sekundäre Engpaß 165
7.1.5 Maßnahmen während und am Ende der Messung 166
7.2 Bewertung weiterer Komponenten 167
7.2.1 Grenzdurchsatz beim Filtern 167
7.2.2 Grenzdurchsatz des Paketträgers 169
7.3 Leistungsprognose 171
7.3.1 Anzahl der Prozessoren für Ethernet-Analyse 171
7.3.2 Anzahl der Prozessoren für FDDI-Analyse 172

8 Zusammenfassung und Ausblick 174
8.1 Zusammenfassung der Ergebnisse 174
8.1.1 Ein Rückblick auf die Diplomarbeit 174
8.1.2 Die wichtigsten Ergebnisse 175
8.2 Erfahrungen bei der Realisierung 176
8.2.1 Probleme mit der Hardware 176
8.2.2 Probleme bei der Programmierung 177
8.2.3 Gruppenarbeit 178
8.3 Offene Fragen 178
8.3.1 Die Schnittstelle für Messungen in Hochgeschwindigkeitsnetzen 178
8.3.2 Neue Protokolle mit Verschlüsselungsmechanismen 180

Literatur **182**

Tabelle der Abkürzungen **186**

Abbildungsverzeichnis

2.1 Meßgrößen der physikalischen Schicht 27

3.1 Klassifikation der Zuordnung 40
3.2 Klassifikation der Zuordnungsziele 41
3.3 Blockschaltbild 45
3.4 Zustandsdiagramm 46
3.5 Zeitgeber 48
3.6 Link-Protokoll 49
3.7 'MTM-PC-Board' 50
3.8 'INMOS-kompatibles-Board' 51
3.9 'Hardware-Architektur' 53
3.10 Strukturelle Parallelität 58
3.11 Protokollpipeline 59
3.12 Sende/- Empfangspipeline 59
3.13 Protokollfunktionen 60
3.14 Verbindungsparallelität 61
3.15 Paketparallelität 62

4.1 6-Prozessor-Konfiguration 73
4.2 Direkte und indirekte Messung 76
4.3 Erkennungsbaum für Protokolle 77
4.4 'Deadlock' beim Nachrichtenaustausch 86
4.5 Zugriff einer OS/2 Anwendung auf Geräte 89

5.1 Symbole der Datenflußdiagramme 95
5.2 Datenflußdiagramm der Transputerprozesse 96
5.3 Datenflußdiagramm des Prozeß-Templates 97
5.4 Datenflußdiagramm des Netz-'Servers' 98
5.5 Datenflußdiagramm von 'PDEV' 101
5.6 Datenflußdiagramm von 'PPAKET' 102
5.7 Datenflußdiagramm von 'PSTAT' 103
5.8 Gerätetreiber als 'Clients' und 'Server' 106
5.9 'Template' einer Analysefunktion 109
5.10 'Ethernet- und IEEE 802.3 Frame' 111

5.11 (R)ARP-Dateneinheit 114
5.12 Datagramm (IP-Dateneinheit) 116
5.13 ICMP-Dateneinheit 118
5.14 UDP-Dateneinheit 118
5.15 TCP-Dateneinheit 119
5.16 globale Statistiken 121
5.17 Algorithmus zum Programmieren einer Verbindungstopologie 127
5.18 Topologie "A2B0B2D1D2E0A3C0C2D3" 128
5.19 'P-Code' eines Filterausdrucks 131
5.20 Allgemeines Nachrichtenformat 132
5.21 Vermitteln von Nachrichten 133
5.22 Verteilen von Dateneinheiten 135
5.23 Vereinfachter Zugriff auf Geräte 136
5.24 Empfangen einer Nachricht auf dem 'Hostrechner' 137

6.1 Fenster des 'Hostrechner'-Prozesses 147
6.2 'Standardausgabe'-Fenster 148
6.3 Das 'File'-Menü 149
6.4 Das 'Show'-Menü 150
6.5 Das 'Messung'-Menü 150
6.6 'Standardeingabe'-Fenster 151
6.7 Ein zu kleines Fenster 151
6.8 Protokolle wählen 156
6.9 Zu protokollierende Informationen wählen 157
6.10 ARP/RARP-Fenster 158
6.11 Fenster 'MBit/s' 160

7.1 Grenzdurchsatz in Aufträge/s 163
7.2 Grenzdurchsatz in KByte/s 1/2 164
7.3 Grenzdurchsatz in Aufträge/s 2/2 165
7.4 Grenzdurchsatz in MBit/s 166
7.5 Grenzdurchsatz bei verschiedenen Filter-Aufträgen 168
7.6 Grenzdurchsatz eines Paketträgers 170
7.7 Anzahl Prozessoren für Ethernet-Analyse 172
7.8 Anzahl Prozessoren für FDDI-Analyse 173

8.1 FDDI-LAN 179
8.2 Typisches ATM-LAN 180
8.3 IPv6 Dateneinheit 181

Einleitung und Motivation

An Meßwerkzeuge der Kommunikationsmeßtechnik werden immer wieder neue Anforderungen gestellt. Neben einer Klassifikation dieser Werkzeuge und einer Einordnung der Kommunikationsmeßtechnik als Analysemethode für Kommunikationssysteme wird in Kapitel 1 dargestellt, welchen Anforderungen ein Meßwerkzeug heutzutage genügen muß und welche Anforderungen in Zukunft durch die aufkommenden Hochgeschwindigkeitsnetze dazukommen (vgl. Kapitel 2). Es wird dargestellt, daß die Parallelverarbeitung eine Möglichkeit ist, um den steigenden Anforderungen an die Leistungsfähigkeit zu genügen.

In Kapitel 3 wird neben einer formalen Präzisierung der Parallelverarbeitung mit der Auswahl eines Konzeptes der parallelen Protokollverarbeitung begonnen und begründet, warum sich dieses Konzept für die Realisierung eines parallelen Meßwerkzeuges verwenden läßt. Neben den Bewertungskriterien Leistungsfähigkeit, Skalierbarkeit und Erweiterbarkeit darf bei der Beurteilung eines Konzeptes die Hardware für eine spätere Realisierung nicht vernachlässigt werden. Die für den Prototypen verfügbare Hardware und sich daraus ergebende Restriktionen werden ebenfalls in diesem Abschnitt dargestellt. Am Anfang des Kapitels 4 wird eine Konfiguration auf Grundlage des gewählten Konzeptes vorgestellt, die bereits wesentliche Aspekte des zu realisierenden Prototypen festlegt. Hier wird bereits in groben Zügen festgelegt, welche Funktionen als Prozesse zusammenwirken sollen.

In Kapitel 4 werden die benötigten Funktionen ausführlich beschrieben. Es wird dargestellt, wozu die jeweilige Funktion benötigt wird und wie sie zu klassifizieren ist. In Kapitel 5 wird schließlich beschrieben, wie die Funktionen letztendlich realisiert wurden. Grundlegende Algorithmen werden mit Nassi-Schneiderman Diagrammen verdeutlicht. Die Funktionen werden zu 'Threads' und Prozessen zusammengestellt.

In Kapitel 6 wird der Umgang mit dem Prototypen beschrieben. Dieses Kapitel ist sozusagen Benutzerhandbuch, 'Troubleshooting Guide' und Referenz. In Kapitel 7 werden Engpässe des Prototypen identifiziert und eine Abschätzung der Leistungsfähigkeit gegeben.

Die Zusammenfassung und Ausblick auf weitere Forschungsaktivitäten ist im letzten Kapitel.

Kapitel 1

Analyse von Kommunikationssystemen

Neben einer allgemeinen Definition für *Analyse* wird in diesem Kapitel dargestellt, aus welchen Gründen man ein Kommunikationssystem analysieren möchte (vgl. Abschnitt 1.1) und welche Vorgehensweisen bzw. Analysemethoden grundsätzlich möglich sind (vgl. Abschnitt 1.2).

Es wird vor allem auf die Analyse vorhandener Kommunikationssysteme mit Hilfe von Messungen (vgl. Abschnitt 1.3) eingegangen, um schließlich den Entwurf und die Realisierung eines Werkzeuges für diesen Zweck zu motivieren.

In einem eigenen Abschnitt (vgl. 1.4) werden die verschiedenen Klassifikationen für Werkzeuge der Kommunikationsmeßtechnik vorgestellt.

1.1 Notwendigkeit für die Analyse von Kommunikationssystemen

Mit *Analyse* ist im allgemeinen eine *zielgerichtete, aktiv-empirische Untersuchung* gemeint. Der Zweck einer Analyse ist die Gewinnung nicht offensichtlicher Informationen über ein System bzw. Teilsystem. Neben der *Interpretation* der Analyseergebnisse ist die Gewinnung von Informationen durch eine Analyse die wichtigste Grundlage für die *Bewertung* eines Systems. Im folgenden wird speziell auf Kommunikationssysteme als Ganzes (eventuell sogar als Subsystem eines noch größeren Systems) oder auf Teile eines Kommunikationssystems, wie beispielsweise Vermittlungsrechner, Leitungen oder Protokolle, eingegangen.

Kommunikationssysteme werden in der Regel als Subsysteme entworfen und betrieben. Als Teil eines größeren Systems, beispielsweise eines "Verteilten Systems", werden eine Vielzahl an (Benutzer-) Anforderungen an das Kommunikationssystem gestellt. Dies sind insbesondere:

- Verfügbarkeit: Ein System muß grundsätzlich verfügbar sein, um überhaupt einen Nutzen erbringen zu können. Bei einer Analyse soll in der Regel fest-

gestellt werden, wie das zeitliche Verhältnis zwischen verfügbarem System und nicht-verfügbarem bzw. teilweise verfügbarem System ist[1].

- Leistungsfähigkeit: Komplexere Anforderungen der Benutzer benötigen immer leistungsfähigere Kommunikations- und Rechensysteme. Bei der Analyse der Leistungsfähigkeit soll in der Regel festgestellt werden, welchen Durchsatz ein System bei der Abarbeitung verschiedener Aufträge hat.
- Zuverlässigkeit: Die Kommunikation sollte zuverlässig sein. Hiermit ist insbesondere "fehlerfrei" gemeint.
- Erweiterbarkeit: Durch neue Aufgaben oder höhere Leistungsanforderungen kann es nötig werden, ein Kommunikationssystem zu erweitern. Beispielsweise müssen neue Segmente in ein vorhandenes Netz aufgenommen werden, um weitere Endgeräte anzuschließen. Auch die Integration neuer oder geänderter Protokolle ist ein Aspekt der Erweiterbarkeit.

Die quantifizierbaren (Benutzer-) Anforderungen eignen sich besonders gut als Bewertungskriterien für Kommunikationssysteme. Anhand quantifizierbarer Größen wie Verfügbarkeit und Leistungsfähigkeit, können verschiedene (geplante) Systeme miteinander verglichen werden (vgl. [Wol93, S. 5]).

Die Analyse — als Grundlage für die Bewertung eines Kommunikationssystems — wird benötigt, um entscheiden zu können,

- ob die Anforderungen der Benutzer von einem geplanten/vorhandenen Kommunikationssystem erfüllt werden können bzw.
- wie ein Kommunikationssystem aussehen müßte, um die Anforderungen zu erfüllen,
- wie man ein vorhandenes Kommunikationssystem an neue/andere Anforderungen anpassen kann,
- wie man Kommunikationssysteme möglichst optimal konfiguriert[2],
- ob ein Kommunikationssystem sich im laufenden Betrieb seiner Spezifikation entsprechend verhält[3],
- ob in das Kommunikationssystem von außen eingebrochen wird,
- ob es zu Verklemmungen bei der Betriebsmittelverwaltung kommt,
- wer welche Betriebsmittel wie lange benutzt.

[1] In der Regel sind ausfallanfällige Systeme bei Wartungen oder Reparaturen nicht oder nur teilweise verfügbar.

[2] Beim Konfigurieren legt man Systemparameter fest. Hierbei kann es sich beispielsweise um die zu verwendenden Protokolle handeln, oder um die Erstellung statischer Wegwahltabellen für Vermittlungsrechner.

[3] Ein Beispiel ist das Testen von neuen Protokollen. Bei der Spezifikation können in der Regel formale Validierungstechniken eingesetzt werden. Bei einer anschließenden manuellen Implementation kann es dennoch zu Fehlern kommen, die dann im laufenden (Test-) Betrieb gesucht werden.

1.2 Analysemethoden

Grundsätzlich kann man zwischen direkten Analysen am Kommunikationssystem und indirekten Analysen an Modellen unterscheiden. Die direkte Analyse setzt voraus, daß das zu untersuchende Kommunikationssystem bereits existiert. In diesem Fall kann man mit Hilfe von Meßinstrumenten Informationen direkt aus dem Kommunikationssystem gewinnen und auswerten. Für die Planungs- und Entwurfsphase eines Kommunikationssystems ist man grundsätzlich auf die modellgestützte Analyse angewiesen, da in diesen Stadien das zu analysierende System nicht existiert (vgl. [Wol93, S. 1]).

Die modellgestützte Analyse kann wiederum simulativ oder mathematisch-analytisch erfolgen. Auch die Kombination beider Verfahren ist möglich. Voraussetzung für eine modellbasierte Analyse ist in jedem Fall die Erstellung und Validierung eines Modells (vgl. [Pag91, S. 13ff]).

Ideal wäre für die Durchführung einer Analyse, die geeigneteste Analysemethode für den jeweiligen Fall (d.h. für das Ziel der Untersuchung) zu wählen.

In der Praxis ist die Wahl einer Analysemethode jedoch auch von anderen Faktoren abhängig. Beispiele hierfür sind:

Mangelhafte Kenntnisse/Fähigkeiten des Analysierenden

In der Regel benötigt man für die mathematisch-analytische Methode umfassende mathematische Kenntnisse, da bereits kleine Modelle eine sehr komplexe formale Beschreibung haben können. Darüber hinaus gibt es mehrere verschiedene formale Beschreibungstechniken für die Modellierung eines Systems[4]. Der Analysierende wählt für die Beschreibung meist den Formalismus, den er selber am besten beherrscht. Die Eignung des gewählten Formalismus für die durchzuführende Untersuchung sollte jedoch das primäre Kriterium für die Auswahl sein.

Mangelhafte Werkzeugunterstützung

Für alle Analysemethoden gibt es Werkzeuge, die den Analysierenden bei einer Vielzahl seiner Tätigkeiten unterstützen. Derartige Werkzeuge sind aufgrund ihrer Komplexität jedoch meist nicht intuitiv anwendbar, so daß eine Einarbeitungszeit nötig ist. Auch hier ist jedoch das primäre Problem, daß eine bestimmte Analysemethode nur deshalb gewählt wird, weil für sie ein Werkzeug verfügbar ist.

Zeitmangel

Müssen Analysen innerhalb eines Projektes termingerecht abgeliefert werden, kann der Termindruck dazu führen, daß der Analysierende die Methode wählt, die (vermeintlich) am schnellsten zu einem Ergebnis führt. Simulationsstudien an einem Modell

[4] Beispiele hierfür sind "Markov Ketten", "Fehler-Baum-Diagramme" oder "Stochastische Petri Netze". Einen vergleichenden Überblick findet man in [MWT94]

sind meist besonders zeitaufwendig. Erstens muß ein Modell erstellt (i.d.R. in einer Programmiersprache programmiert) werden. Das ist zeitaufwendig, da neben der Programmierung auch die Validierung des Modells und eventuell anschließende Modifizierungen vorgenommen werden müssen. Zweitens sind die Experimente am Modell zeitaufwendig. Der Zeitaufwand für die Experimente ist u.a. abhängig vom Untersuchungsziel. Bei Optimierungsmodellen muß die Analyse mehrmals mit verschiedenen Parametern durchgeführt werden, bis eine (sub)optimale Parametermatrix gefunden ist (vgl. [Wol93, S. 9]). Bei anderen Modellen muß i.d.R. das gleiche Experiment mehrmals durchgeführt werden, um die statistischen Berechnungen abzusichern.

Der gewünschte Abstraktionsgrad.

Die Entscheidung, auf welchem Abstraktionsgrad eine Analyse durchgeführt werden soll, ist ein weiteres Kriterium für die Wahl der Methode. Meist läßt sich ein mathematisches Modell nicht mit dem gleichen Detaillierungsgrad aufstellen wie ein vergleichbares Modell für ein Simulationsexperiment.

Weitere Faktoren für oder gegen die Wahl einer bestimmten Analysemethode werden in [Jai91, S. 30ff] diskutiert.

1.3 Messungen am Kommunikationssystem

Direkte Analysen am Kommunikationssystem beginnen mit dem Erheben von Daten. Neben statischen Angaben zum Kommunikationssystem, wie z.B. maximale Übertragungsgeschwindigkeit einer Netzkomponente, sind dies vor allem Meßdaten, die dynamisch durch Messungen am Kommunikationssystem anfallen.

In diesem Abschnitt soll dargestellt werden, warum Messungen erforderlich sind (vgl. Abschnitt 1.3.1) und welche Werkzeuge man für Messungen an Kommunikationssystemen einsetzt. Im folgenden Abschnitt (1.4) wird eine Klassifikation dieser Werkzeuge vorgestellt.

1.3.1 Die Notwendigkeit für Messungen am Kommunikationssystem

Für Netzverwalter ergeben sich drei große Aufgabenbereiche während des laufenden Betriebes, bei denen Messungen unverzichtbar sind. Dies sind:

- Erkennen von Fehlern,
- Erkennen der Belastung des Kommunikationssystems,
- Erkennen von Sicherheitsrisiken.

Neben diesen Aufgaben (s.u.), die zum Netzmanagement gehören, können Messungen bei der Systemanalyse benutzt werden, um

- Schätzwerte für Modellparameter durch empirische Daten zu ersetzen und
- Ergebnisse von Modellexperimenten mit empirischen Daten zu vergleichen (z.B. zum Zwecke der Modellvalidierung).

1.3.1.1 Erkennen von Fehlern

Das Erkennen von Fehlern ist offensichtlich notwendig, um den störungsfreien und sicheren Betrieb des Kommunikationssystems zu gewährleisten. Die häufigsten Fehlerursachen in Kommunikationssystemen sind (vgl. [GK94, S. 331f])

- fehlerhafte oder mehrdeutige Spezifikationen einzelner Komponenten,
- höhere Verkehrslasten im Betrieb als in der Planung,
- Fehler bei der Realisierung der Komponenten ('Hardwarefehler' und 'Softwarefehler'),
- fehlerhafte Installation oder Konfiguration,
- Ausfall von Komponenten.

Der Fehler ist um so schwerwiegender, je höher er in der Liste steht. Bei einem Ausfall von Komponenten kann man den Fehler durch Austauschen der betreffenden Komponenten beseitigen. Eine Anpassung des Kommunikationssystems an höhere Verkehrslasten ist wesentlich umfangreicher, da Teile oder auch das ganze System neu strukturiert und erweitert werden müssen. Stellen sich während des Betriebs Spezifikationsfehler heraus, erfordert dies in der Regel ein aufwendiges (weil nachträgliches) 'Neuspezifizieren' mit anschließender Entwicklung, Realisierung, Konfigurierung und Inbetriebnahme der betroffenen Komponenten.

Das Erkennen von potentiellen Fehlern ist ebenfalls wichtig. So kann etwa die Lastmessung (s.u.) benutzt werden, um mögliche Fehler aufgrund von zu hoher Last an einer Komponente frühzeitig aufzufinden. Der Netzverwalter kann dann geeignete Gegenmaßnahmen treffen, bevor es zu Fehlern kommt.

Auch das Messen von Dienstgüte ('Quality of Service') gehört in diese Kategorie. Es ist i.d.R. 'Verhandlungssache' zweier Partnerinstanzen, welche Dienstgüte für eine Übertragung gelten soll. Dennoch kann es bei einer zu hohen 'Bitfehlerrate' einer Leitung zu Abbrüchen von Übertragungen kommen, was zumindest aus Anwendersicht als Fehler zu klassifizieren ist. Messungen können auch in diesem Fall helfen, eine bestimmte Dienstgüte nachzuweisen.

1.3.1.2 Erkennen der Belastung eines Kommunikationssystems

Das Erkennen der Belastung eines Kommunikationssystems ist notwendig, um entscheiden zu können, ob und wie das System zu rekonfigurieren oder zu erweitern ist. Insbesondere sind hier zu nennen:

- Betriebsmittelauslastung ('Accounting'),
- Verkehrs- und Lastanalysen,
- Messen des Durchsatzes.

Eine Analyse der Betriebsmittelauslastung zeigt schnell auf, wo es zu Engpässen einerseits und unnötiger Redundanz andererseits kommt. Zeigen beispielsweise die 'Accounting-Einträge' für Drucker in einem lokalen Netz, daß vor allem die Drucker in unmittelbarer Nähe zu den Arbeitsplatzrechnern benutzt werden, so könnte dies Grundlage für die Entscheidung sein, zentrale 'Druckerpools' aufzulösen. Die so frei werdenden Drucker könnten dann auf die Räume mit den Arbeitsplatzrechnern aufgeteilt werden.

Verkehrs- und Lastanalysen zeigen schnell potentielle Engpässe in einem Kommunikationssystem auf. Wenn beispielsweise die Verkehrsanalyse in einem stark beanspruchten Teil eines Kommunikationssystems ergibt, daß bestimmte Gruppen von Komponenten vorwiegend untereinander kommunizieren, so könnte man die Last in diesem Teil des Kommunikationssystems verringern, indem man durch 'Brücken' oder Vermittlungsrechner die verschiedenen Gruppen voneinander trennt.

Das Messen des Durchsatzes soll vor allem zeigen, welche Klassen von Aufträgen ein System besonders belasten. Derartige Messungen können Entscheidungsgrundlage für die Beschaffung von neuen Komponenten sein, die bestimmte Klassen von Aufträgen schneller oder auch sicherer bearbeiten.

1.3.1.3 Erkennen von Sicherheitsrisiken

Das Erkennen von Sicherheitsrisiken ist notwendig, um frühzeitig Maßnahmen zum Schutz der Anwender und deren Daten ergreifen zu können. Gerade 'offene' Systeme bieten durch ihre Vernetzung in Kommunikationssystemen eine Vielzahl an Einbruchsmöglichkeiten. Neben der Möglichkeit das lokale Netz von überregionalen Netzen abzuschotten bzw. die Kommunikation zwischen den Netzen zu reglementieren[5], bieten Messungen die Möglichkeit

- potentielle Sicherheitslücken aufzuzeigen,
- den Zugriff auf sensible Daten zu registrieren,
- Verstöße gegen Sicherheitsmaßnahmen zu erkennen.

1.3.1.4 Aufdecken potentieller Sicherheitslücken

Potentielle Sicherheitslücken bilden Rechner, die als Server Dienste über ein Netz anbieten. Bei einer zu freizügigen oder einfach falschen Konfiguration könnten Benutzer über ein Netz den Betrieb des Servers bewußt oder unbewußt stören. Durch Messungen, die Konfigurationsfehler aufdecken, können geeignete Gegenmaßnahmen veranlaßt werden (vgl. z.B. [Kus95]).

[5] Sogenannte 'Firewalls' (vgl. [Kie95]) übernehmen die Abschottung des lokalen Netzes nach außen. Diese Technik hat jedoch eine Reihe von Nachteilen. Insbesondere ist hier zu nennen, daß dedizierte Rechner für diese Aufgabe bereitgestellt werden müssen, die potentiell Engpaß der gesamten Kommunikation in einem LAN werden können. Gerade weil die 'Firewalls' die Kommunikation von und nach 'außen' überwachen, muß die gesamte LAN-WAN-Kommunikation über sie ablaufen.

1.3.1.5 Zugriff auf sensible Daten

Der Zugriff auf sensible Daten kann analog zum 'Accounting' (s.o.) gemessen werden. Dieses Vorgehen ist aus Datenschutzgründen selber nicht unproblematisch. Dennoch kann so registriert werden, wann, von wo und wie lange auf sensible Daten zugegriffen wird.

1.3.1.6 Verstöße gegen Sicherheitsmaßnahmen

Verstöße gegen Sicherheitsmaßnahmen können ebenfalls durch Messungen erkannt werden. Kann man beispielsweise durch die Vergabe oder den Entzug von Zugriffsrechten nicht sicherstellen, daß bestimmte Personen(gruppen) einige Dienste oder Betriebsmittel nicht benutzen, da so neue Sicherheitslücken (s.o.) entstehen, ist es notwendig derartige Verstöße zu registrieren. So kann schnellstmöglich nach einem Verstoß eine geeignete Maßnahme veranlaßt werden. Beispielsweise könnte der Verantwortliche für ein 'Verteiltes System' darauf bestehen, daß alle Systemverwalter mit bestimmten Zugriffsrechten das bekannte 'TELNET'-Protokoll nicht benutzen dürfen, um sich über das Kommunikationssystem auf einem anderen Rechner einzutragen. Durch einfaches Abhören der jeweiligen Verbindung können Unbefugte so in Kenntnis von 'Password' und Benutzernamen kommen.

Durch einfaches Abhören der Verbindungen kann aber auch ein Meßwerkzeug dazu benutzt werden, um derartige Verstöße gegen das 'TELNET-Verbot' zu überwachen. Eine geeignete Gegenmaßnahme wäre das sofortige Ändern des potentiell bekannten 'Password' durch den Benutzer[6].

1.3.2 Werkzeuge zum Erfassen von Meßdaten

Im Bereich der Kommunikationsmeßtechnik gibt es eine Vielzahl von Werkzeugen zum Erfassen der verschiedenartigen Meßdaten im Kommunikationssystem. Diese Werkzeuge werden *Monitore* genannt.

In der Informatik gibt es noch weitere Definitionen für Monitor. Teilweise werden die Bildschirme der Rechner als 'Monitore' bezeichnet. Im Zusammenhang mit Betriebssystemen versteht man unter einem 'Monitor' eine Funktion, die den Zugriff auf Betriebsmittel für nebenläufige Prozesse synchronisiert.

In dieser Arbeit wird der Begriff 'Monitor' ausschließlich als Bezeichnung für *ein Werkzeug zum Erfassen von Meßdaten* benutzt. Über den Funktionsumfang, was ein Monitor alles 'können' muß, gibt es unterschiedliche Meinungen. Grundvoraussetzung für einen Monitor ist die Fähigkeit zur *Datenerfassung* von Rohdaten[7]. Darüber hinaus kann ein Monitor zusätzliche Funktionen beinhalten. Dies sind Funktionen zum:

- Filtern der Daten
- Analysieren der Daten

[6] Diese Änderung sollte jedoch lokal am jeweiligen Arbeitsplatzrechner erfolgen. Eine Änderung mit Hilfe von 'TELNET' würde das Sicherheitsrisiko nicht beseitigen.

[7] Rohdaten sind die vom Monitor unmittelbar am System erhobenen Meßdaten.

- Bewerten der Daten
- Anzeigen der Daten
- Verwalten der Daten
- aktivem Testen des Meßobjektes[8]

1.3.2.1 Filtern von Daten

Das Filtern der Daten ist erforderlich, wenn der Monitor nur einen Teil der gemessenen Daten weiterverarbeiten soll oder — aufgrund des Betriebsmittelbedarfs für die Weiterverarbeitung der Daten — nur einen Teil der gemessenen Daten weiterverarbeiten kann. Da der Betriebsmittelbedarf für Monitore in der Regel sehr hoch ist, sollte das Filtern der Rohdaten möglichst unmittelbar nach der Erfassung erfolgen.

1.3.2.2 Analysieren der Daten

Unter Analysieren ist in diesem Zusammenhang das zielgerichtete Auswerten der Rohdaten zu verstehen. Das können statistische Berechnungen sein, Klassifizierung der Meßdaten etc.

1.3.2.3 Bewerten der Daten

Beim Bewerten der Daten werden Analyseergebnisse vom Monitor interpretiert. Einige Werkzeuge zur Fehlersuche bieten sogar die Funktionalität von Expertensystemen. Durch Einsatz von KI-Techniken versucht der Monitor nach dem Analysieren der Meßdaten Rückschlüsse auf die Fehlerursache zu ziehen.

Nicht in jedem Fall sind für eine maschinelle Bewertung von Analyseergebnissen KI-Techniken erforderlich. Oft ergeben sich Bewertungsmöglichkeiten direkt aus den Analyseergebnissen.

1.3.2.4 Anzeigen der Daten

Das Anzeigen der Daten ermöglicht dem Analysierenden den Einblick in die Meßergebnisse. Grundsätzlich kann man zwischen der tabellarischen und der graphischen Aufbereitung unterscheiden.

Bei der tabellarischen Aufbereitung werden die analysierten Meßdaten formatiert dargestellt. Diese Art der Darstellung eignet sich besonders für Protokollanalysatoren (s.u.). Die graphische Aufbereitung ist besonders für große Datenmengen unerläßlich. Durch Histogramme und andere Diagrammformen können die Analyseergebnisse übersichtlich und kompakt dargestellt werden. Auch die Darstellung sich schnell ändernder dynamischer Prozesse ist in der Regel nur graphisch möglich.

[8] In diesem Fall ist das 'Meßobjekt' immer ein Teil des Kommunikationssystems.

1.3.2.5 Verwalten der Daten

Das Verwalten der Daten ist erforderlich, wenn auf sie über längere Zeit zugegriffen werden soll. Bei Messungen, die sehr große Datenmengen in kurzer Zeit erheben, ist trotz Filtern (s.o.) meist keine Analyse und Anzeige der Daten in Echtzeit[9] möglich. Daher ist es nötig, die anfallenden Daten auf Massenspeicher für eine spätere Weiterverarbeitung zwischenzulagern.

In 'Verteilten Systemen' können Messungen parallel an verschiedenen Stellen (Subsystemen) erfolgen. Zum Verwalten der Daten gehört in diesem Fall auch, die dezentral anfallenden Daten lokal für eine Auswertung bereitzustellen. Hierzu gehört beispielsweise das Zusammenführen verschiedener Meßreihen durch Sortieren der erfaßten Daten. Ein weiterer Aspekt der Datenverwaltung ist eine Zugriffskontrolle bei sensiblen Daten.

1.3.2.6 Aktives Testen

Aktives Testen von Komponenten durch einen Monitor kann für verschiedene Meßaufgaben eingesetzt werden. Beispielsweise kann durch einen 'Lastgenerator' eine spezifische Last für das Kommunikationssystem für 'Lastmessungen' (s.u.) generiert werden. Bei 'Protokolltestern' (s.u.) simuliert der Monitor das Verhalten einer Protokollinstanz, um die Partnerinstanz im Kommunikationssystem zu testen.

1.3.2.7 Eine alternative Benennung von Meßwerkzeugen

Aufgrund der vielfältigen Verwendung des Begriffs 'Monitor' für Meßwerkzeuge mit erheblich unterschiedlichem Funktionsumfang, wird in [Wol93, S. 16] eine andere Terminologie vorgeschlagen. Ausgegangen wird von der Grundfunktionalität eines Monitors zur Datenerhebung, um ein 'Meßinstrument' zu definieren. Durch sukzessive Erweiterung der Funktionalität durch Komponenten zur

- Datenreduktion,
- Datenanzeige,
- Analyse,
- Diagnose,
- Netzadministration

erhält man letztendlich über mehrere wohldefinierte Zwischenstufen ein (Performance-) Managementsystem, daß alle bisher beschriebenen Funktionen umfaßt (vgl. Abschnitt 1.4).

[9] In diesem Zusammenhang wird 'in Echtzeit' als parallel zur Datenerfassung verstanden.

1.4 Klassifikation von Monitoren

Eine Klassifikation von Monitoren ist unter verschiedenen Aspekten möglich. Die verschiedenen Beziehungen und Wechselwirkungen der Klassifikationen untereinander werden in diesem Abschnitt aufgezeigt.

Grundsätzlich lassen sich Monitore klassifizieren nach

- der Meßaufgabe,
- ihrer Architektur,
- ihrem Funktionsumfang,
- dem eingesetzten Meßverfahren und
- der Ergebnispräsentation.

1.4.1 Klassifikation nach der Meßaufgabe

Setzt man als Analysemethode die Messung am Kommunikationssystem (vgl. Abschnitt 1.2) ein, so kann man wiederum verschiedene Meßaufgaben unterscheiden, die sich meist direkt aus dem Analyseziel (vgl. Abschnitt 1.1) ableiten lassen. In [GK94, S. 333] wird eine Einteilung der Meßaufgaben in vier Bereiche vorgeschlagen:

- *Übertragungstechnische Messungen an Übertragungswegen.* Gemessen werden analoge Größen wie Impedanzen, Geräusche und Frequenzgang. Bei digitalen Messungen können beispielsweise Bitfehlerhäufigkeiten ermittelt werden[10].
- *Schnittstellentest.* Bei diesen Messungen sollen Kabeldurchgänge, Installationen und Pegelbereiche überprüft werden. Grundlage dieser Tests sind die Normen für die zu vermessenden Schnittstellen.
- *Daten- und Protokollanalyse.* Ziel dieser Messungen ist ein Aufzeichnen und Auswerten des Datenstroms und ein Testen der eingesetzten Protokolle. Durch einen 'Protokollanalysator' kann beispielsweise überprüft werden, ob sich ein Protokoll gemäß seiner Spezifikation verhält.
- *Netzanalyse.* Bei der Netzanalyse wird der 'Durchsatz' und die 'Verkehrslast' in einem Netz gemessen.

Die für die jeweiligen Messungen eingesetzten Monitore leiten sich bei dieser Klassifikation direkt aus dem Namen der Messung ab. So werden bei [GK94] Bitfehlermeßgeräte, Schnittstellentester, Daten-, Protokoll- und Netzanalysatoren unterschieden (vgl. [GK94, S. 337–368]).

Eine genauere Unterscheidung der oben als 'Netzanalyse' klassifizierten Messungen wird in [Wol93, S. 16f] vorgenommen. Ein *Netzmonitor* wird hier subklassifiziert in

[10] Diese Kategorie ist wiederum subklassifizierbar als 'Übertragungstechnische Messungen an analogen Übertragungswegen' und 'Übertragungstechnische Messungen an digitalen Übertragungswegen'.

- LAN-Monitor,
- Stations-Monitor,
- Leitungs-Monitor und
- Protokoll-Monitor.

Diese Terminologie geht unmittelbar auf die zu analysierenden Subsysteme eines Kommunikationssystems ein. Man kann das Kommunikationssystem als ganzes betrachten (LAN-Monitor) oder einzelne Subsysteme: Die Arbeitsplatzrechner, die 'Server' und andere individuell adressierbare Hardware im Netz (Stations-Monitor), deren physikalischen Verbindungen untereinander (Leitungs-Monitor) und die von ihnen eingesetzten Protokolle (Protokoll-Monitor).

Hinzu kommen bei dieser Klassifizierung Lastmonitore für Lastmessungen auf verschiedenen Ebenen des Kommunikationssystems, zur Erstellung von Verkehrsmatrixen und Leistungsmonitore, die im wesentlichen den Durchsatz bestimmter Aufträge bei einer gegebenen Last messen.

Abschließend ist zu der Klassifizierung nach der Meßaufgabe anzumerken, daß sie einerseits intuitiv ist, da sie sich direkt auf die Hauptaufgabe des jeweiligen Monitors bezieht. Andererseits ist sie zwangsläufig unvollständig, da mit neuen Anforderungen und Aufgaben neue Klassen denkbar sind. Darüber hinaus fällt auf, daß auch hier die Namensgebung uneinheitlich ist (vgl. Protokollanalyse vs. Netzanalyse bei [GK94] (s.o.) bzw. Protokoll-Monitor als Subtyp von Netzmonitor bei [Wol93] (s.o.)).

1.4.2 Klassifikation nach der Architektur

Die Unterscheidung von Monitoren nach der Art ihrer Realisierung bzw. Implementierung ist die am weitesten verbreitete Klassifikation. Es kann grundsätzlich unterschieden werden zwischen

- Hardware-Monitor,
- Software-Monitor,
- 'Firmware-Monitor' und
- 'Hybrid-Monitor'.

Obwohl der Trend bei neueren Monitoren zu einer hybriden Realisierung geht, ist diese Klassifizierung weiterhin breit akzeptiert (vgl. z.B. [Hin86], [Jai91], [Thu93]). Dies liegt vor allem daran, daß *"in der Anfangszeit der Monitortechnik [...] man fast jeden Monitor genau einer von zwei Klassen zuordnen konnte. Er war entweder ein Hardware- oder ein Software-Monitor"* (vgl. [Hin86, S. 4]).

1.4.2.1 Hardware-Monitor

Ein Hardware-Monitor ist ein Meßgerät, das an das Meßobjekt[11] mit Hilfe von Meßfühlern angeschlossen wird. Veränderungen des Meßobjektes werden durch die Meßfühler registriert. Die Meßergebnisse können durch logische Schaltungen weiterverarbeitet werden. Einige Hardware-Monitore unterstützen auch das Speichern der Daten auf eigenen Massenspeichern (vgl. [Lip88, S. 31], [Hin86, S. 4ff], [Jai91, S. 98]).

Wesentliches Merkmal der Hardware-Monitore ist, daß sie das Verhalten des Meßobjektes nicht beeinflussen. Insbesondere das zeitliche Verhalten der Zustandsänderungen des Meßobjektes ist unabhängig davon, ob ein Hardware-Monitor angeschlossen ist oder nicht. Weitere Vorteile von HW-Monitoren (HW-Monitor) sind:

- Es sind sehr hohe Abtastraten möglich, meist um mehrere Größenordnungen höher als bei vergleichbaren Software-Monitoren.
- Es werden keine Betriebsmittel des untersuchten Systems beansprucht.
- Es können durch den Einsatz von HW-Monitoren kaum Fehler in das zu untersuchende System eingebracht werden.
- Es sind hardwarenahe Messungen möglich.

Der Hauptnachteil eines HW-Monitors ist, daß er nur solche Veränderungen (oder Zustände) des Meßobjektes registrieren kann, die durch die Meßfühler zugänglich sind.

1.4.2.2 Software-Monitor

Ein Software-Monitor ist ein Programm, das Teil des Meßobjektes ist bzw. mit dem Meßobjekt zusammen auf derselben Hardware abläuft (vgl. [Hin86, S. 6f], [Lip88, S. 32]). Im Gegensatz zum HW-Monitor beeinflußt ein Software-Monitor (SW-Monitor) i.d.R. das zeitliche Verhalten des Meßobjektes. Wenn man die Störungen des Meßobjektes durch den SW-Monitor bei der Analyse nicht berücksichtigt, kann das zu einer Verfälschung der Meßergebnisse führen. Dies ist der wesentliche Nachteil aller SW-Monitore. Die Vorteile der SW-Monitore sind:

- Sie sind einfacher realisierbar als HW-Monitore.
- Sie sind leicht veränderbar/erweiterbar.
- Es sind umfangreiche Analysen der Meßwerte möglich, da die Funktionalität des Meßobjektes für diese Analysen meist vollständig zur Verfügung steht.
- Es sind applikationsnahe Messungen möglich.

SW-Monitore lassen sich subklassifizieren nach der Art, wie Veränderungen des Meßobjektes registriert werden. Die Aufnahme von Meßdaten durch den Monitor kann vom Meßobjekt durch folgende drei Möglichkeiten veranlaßt werden (vgl. [Jai91, S. 96]):

[11] Mit 'Meßobjekt' ist das zu vermessene System gemeint.

- Ausführen bestimmter Maschinenbefehle.[12]
- Ausführen des Einzelschrittmodus. Der Prozessor des Meßobjektes oder der Hardware, auf dem Monitor und Meßobjekt ablaufen, springt nach Ausführung jedes einzelnen Maschinenbefehls in den 'Programmcode' des Monitors. Der Einzelschrittmodus[13] muß vom jeweiligen Prozessor unterstützt werden.
- Unterbrechungen durch den Zeitgeber. Verfügt das Meßobjekt über einen Zeitgeber, dann kann der Monitor durch eine vom Zeitgeber ausgelöste Unterbrechung zur Aufnahme von Meßdaten veranlaßt werden.

Diese Subklassifikation ist außerdem eine Verfeinerung der Klassifikation nach dem Meßverfahren (s.u.).

1.4.2.3 Vergleich von Hardware- und Software-Monitoren

Die Vorteile bzw. Nachteile von HW- und SW-Monitore ergeben sich im direkten Vergleich beider Typen miteinander. Neben den oben genannten Hauptnachteilen beider Typen haben die SW-Monitore weitere Nachteile.

Aufgrund der engen Koppelung der SW-Monitore mit dem Meßobjekt, kann ein SW-Monitor fehlerhafte Ergebnisse produzieren, wenn das beobachtete System selber fehlerhaft ist. Ein HW-Monitor liefert hingegen auch von fehlerhaften Systemen korrekte Meßwerte. Ist das System ausgefallen, steht der SW-Monitor überhaupt nicht mehr zur Verfügung.

Die enge Koppelung mit dem System verhindert i.d.R. auch den Einsatz von SW-Monitoren auf anderen Systemen. Falls dies überhaupt möglich ist, müssen Anpassungen an andere Systeme vorgenommen werden[14].

Aufgrund ihrer anwendungsnahen Einbindung in das zu untersuchende System, werden parallel auftretende Ereignisse[15] meist sequentiell vom SW-Monitor aufgenommen. Ein Beispiel sind Unterbrechungssignale ('Interrupts') eines Rechners. Diese können parallel auftreten, werden durch die 'Rechnerhardware' jedoch sequentiell zur Abarbeitung an das Betriebssystem weitergegeben. Ein SW-Monitor kann frühestens auf dieser Ebene (Betriebssystem) die Unterbrechungen registrieren, nimmt sie also in jedem Fall als Sequenz auf.

Ein HW-Monitor könnte durch mehrere Meßfühler an den Unterbrechungsleitungen des Rechners parallel auftretende Unterbrechungen auch als solche registrieren.

HW-Monitore hingegen sind teuer. Erweiterungen oder Veränderungen sind meist mit erheblichem Aufwand verbunden, sofern dies überhaupt möglich ist. Der Vorteil der 'hardwarenahen' Messung wird schnell wieder durch die geringe Aussagekraft der

[12] Dies sind sogenannte 'Trap Instructions'. I.d.R. wird der Programmzähler und somit die Ausführung des Programms durch 'Trap Instructions' auf den Monitor gesetzt. 'Trap Instructions' sind abhängig vom Prozessortyp.

[13] engl. 'Trace Mode'

[14] Beispielsweise eine erneute Übersetzung der Programmquellen des SW-Monitors auf dem zu untersuchenden System.

[15] Der Begriff 'Ereignis' wird weiter unten eingeführt.

Meßergebnisse ausgeglichen. Beispielsweise kann ein HW-Monitor gut den Durchsatz in Bit/s einer Leitung messen. Das Erkennen der Struktur der übertragenen Informationen leisten HW-Monitore jedoch nicht.

Für eine weitere Diskussion siehe [Jai91, S. 98ff].

1.4.2.4 'Firmware-Monitor'

Ein 'Firmware-Monitor' ist Teil des 'Mikrocodes' eines Prozessors. Seine Funktion beschränkt sich meist darauf, die Häufigkeit für das Auftreten bestimmter Maschinenbefehle zu zählen. Auf 'Firmware-Monitore' wird nicht weiter eingegangen.

1.4.2.5 'Hybrid-Monitor'

Ein 'Hybrid-Monitor' ist eine Kombination aus HW- und SW-Monitor. Man versucht durch einen Hybrid-Monitor die Vorteile von HW- und SW-Monitor zu vereinen (vgl. [Lip88, S. 33]. Gängige Realisierungen benutzen spezielle Hardware für den Empfang und die Auswertung der Meßdaten sowie für Schnittstellen am Meßobjekt, über die *programmgesteuert* die Meßdaten zum 'Hardware-Teil' gesendet werden. Die programmgesteuerte Datenerfassung nutzt die Vorteile der SW-Monitore. Durch die externe Weiterverarbeitung der Meßwerte wird jedoch die Beeinflussung des Meßobjektes minimiert.

1.4.3 Klassifikation nach dem Funktionsumfang

Diese Klassifikationsmöglichkeit wurde in Abschnitt 1.3.2.7 kurz angesprochen. Es handelt sich bei dieser Klassifikation um den Versuch, wesentliche Funktionsmerkmale zusammenzufassen. Im Gegensatz zur Klassifikation nach der Meßaufgabe ergibt sich so eine abgeschlossene Einteilung der Werkzeuge (vgl. [Wol93, S. 16]):

- *Meßinstrument*: Ein Werkzeug mit Komponenten zur Datenerfassung und Datenreduktion.
- *Meßdatenerfassungsgerät*: Ein Meßinstrument erweitert um Funktionen zur Anzeige der Daten.
- *Meßdatenerfassungs- und -analysegerät* (MEAG): Ein Meßdatenerfassungsgerät mit Komponenten zur Datenanalyse.
- *Diagnosegerät*: Ein MEAG erweitert um Diagnose- und Testeinrichtungen.
- *'Performance-Managementsystem'*: Ein MEAG erweitert um Netzadministrationsfunktionen.

Ein Nachteil dieser Klassifikation ist, daß eine strikte Zuordnung eines Monitors zu einer dieser Kategorien diesen eventuell "unterbewertet". Beispielsweise wäre ein Werkzeug mit Funktionen zur Datenerfassung, -reduktion und -analyse kein MEAG, da ihm

die Funktionen zur Anzeige fehlen. Dies ist jedoch bei Werkzeugen, die im Hintergrund arbeiten[16], der Regelfall (s.u.).

1.4.4 Klassifikation nach dem Meßverfahren

Ein Monitor kann Daten entweder

- ereignisgesteuert oder
- zeitgesteuert

erfassen.

Bei ereignisgesteuerten Monitoren ('event-driven') erfolgt die Datenerfassung nur nach dem Auftreten bestimmter Ereignisse im Meßobjekt. Der Vorteil dieses Meßverfahrens ist, daß Daten nur dann erhoben werden, wenn ein vorher definiertes Ereignis eintritt. Besonders bei SW-Monitoren verringert dies die Beeinflussung des Meßobjektes, falls die zu erfassenden Ereignisse selten eintreten. Bei häufig eintretenden Ereignissen produziert dieses Meßverfahren bei SW-Monitoren erheblichen 'Overhead' im beobachteten System. Der Platzbedarf für das Speichern der Daten ist nicht im voraus berechenbar.

Ein zeitgesteuerter Monitor erfaßt die Daten zu bestimmten Zeitpunkten. Dies geschieht meist periodisch nach Verstreichen einer definierten Zeit. Ein Monitor, der seine Daten zeitgesteuert ('sampling') erfaßt, bietet die Möglichkeit, den Betriebsmittelbedarf für die Datenspeicherung genau berechnen zu können. Aus der Abtastfrequenz und der Anzahl der erfaßten Bit pro Abtastung kann der Speicherbedarf errechnet werden, sofern die letztgenannte Datenmenge konstant ist. Ein Nachteil dieser Methode ist, daß Ereignisse, die nicht mit dem Abtastzeitpunkt auftreten, auch nicht erfaßt werden.

Die oben vorgestellte Subklassifikation der SW-Monitore ist auch unter dieser Klassifikation einteilbar. Das Auslösen der Datenerfassung durch *bestimmte Maschinenbefehle* und durch den *Einzelschrittmodus* sind ereignisgesteuerte Meßverfahren. Eine Datenerfassung ausgelöst durch *Zeitgeberunterbrechungen* ist dagegen zeitgesteuert.

1.4.5 Klassifikation nach der Ergebnispräsentation

Diese Klassifikation unterscheidet zwischen

- 'On-line'-Monitoren und
- 'Batch'-Monitoren.

Monitore, die ihre Ergebnisse[17] während der Datenerfassung darstellen, bezeichnet man als 'On-line'-Monitore. 'Batch-Monitore' hingegen sammeln die Meßdaten und

[16] Das ist der sogenannte 'Batch'-Betrieb.

[17] 'Ergebnis' ist hier ganz allgemein zu verstehen. Meßdaten, analysierte Daten, graphische Darstellungen oder Bewertungen des Monitors sind Ergebnisse.

verarbeiten sie zu einem späteren Zeitpunkt bzw. sie überlassen die Analyse und graphische Aufbereitung der Meßdaten (monitor-) externen Werkzeugen.

'On-line'-Monitore kann man für Messungen verwenden, bei denen die Anzeige der Meßergebnisse unmittelbar nach ihrer Erfassung erforderlich ist. Dies ist beispielsweise bei Überwachungsaufgaben wünschenswert, bei denen eine Reaktion auf bestimmte Ereignisse/Meßergebnisse erst nach der visuellen Bewertung eines Verantwortlichen erwünscht ist. 'On-line'-Monitore sind jedoch nur dann verwendbar, wenn die anzuzeigenden Informationen vom menschlichen Betrachter noch während des Anzeigens auswertbar sind. Insbesondere bei der 'Datenanalyse' (s.u.) von Protokollen in Hochgeschwindigkeitsnetzen ist dies nicht mehr möglich.

In Fällen wie diesen, wo die Anzeige der Daten parallel zu ihrer Erfassung den menschlichen Betrachter überfordert, muß man 'Batch'-Monitore verwenden. Sie speichern die Rohdaten oder bereits gefilterte und analysierte Daten auf Massenspeicher für eine spätere ('off-line') Weiterverarbeitung und Anzeige.

1.4.6 Zusammenfassung der Klassifikation

Die oben dargestellten Klassifikationen nach Meßaufgabe, Architektur, Funktionsumfang, Meßverfahren und Ergebnispräsentation können kombiniert werden, um einen Monitor zu beschreiben.

Dies ist erforderlich, da die Einordnung in eine der Klassifikationen meist nicht ausreicht, um ein Werkzeug zu charakterisieren. Außerdem schließen sich die einzelnen Merkmale nicht unbedingt gegenseitig aus. Vielmehr muß man die Merkmale einer Klassifikation als mögliche Freiheitsgrade bei der Wahl eines Werkzeuges verstehen.

So schließt die Möglichkeit eines Monitors Daten 'off-line' zu erfassen nicht aus, daß Teile oder alle Daten bei einer anderen Einstellung auch während der Erfassung angezeigt werden. Ebenso kann ein Monitor die Möglichkeit bieten, sowohl periodisch Daten zu erfassen, als auch auf bestimmte Ereignisse zu reagieren.

Kapitel 2

Anforderungen an ein Meßwerkzeug für Kommunikationssysteme

In diesem Kapitel soll eine möglichst umfassende Darstellung von Anforderungen an ein universelles Meßgerät für Kommunikationssysteme gegeben werden. Die hier dargestellten Anforderungen gehen weit über die Leistungsmerkmale hinaus, die einzelne verfügbare Meßwerkzeuge heute bieten. Dennoch kann man keine vollständige detaillierte Auflistung aller denkbarer Anforderungen aufstellen, da sich ständig neue Anforderungen an Meßwerkzeuge ergeben.

Die Anforderungen an ein universelles Meßgerät für Kommunikationssysteme lassen sich nach den zu erbringenden Funktionen einteilen. Es gibt Anforderungen bezüglich der Datenerfassung (Messung), dem Filtern, Analysieren, Bewerten, Anzeigen, Verwalten der Daten und dem aktiven Testen (vgl. Abschnitt 1.3.2). Außerdem gibt es Anforderungen des Anwenders an ein Werkzeug.

2.1 Anforderungen seitens der Meßaufgabe (Datenerfassung und Analyse)

Die Anforderungen seitens der Meßaufgabe lassen sich in die für Kommunikationssysteme typische Protokollhierarchie einordnen. Die einzelnen Meßaufgaben erfordern Messungen, die auf mindestens einer dieser Protokollschichten stattfindet. Die zu unterstützenden Messungen und darauf aufbauenden Analysen werden anhand der Aufgaben der jeweiligen Protokollschicht motiviert.

2.1.1 Schicht 1, physikalische Schicht

Auf dieser Ebene findet die physikalische Übertragung statt, bei der entweder analoge oder digitale Signale übertragen werden. Folgende Messungen zur Qualität der Leitung und der übertragenen Signale sind gebräuchlich (vgl. auch Abbildung 2.1):

- Impedanzen
- Frequenzgang
- Geräusche
- Jitter (d)
- Spannungsversatz (Bias) (a)
- Gleichspannungsanteil (b)
- Wechselspannungsanteil
- Flankensteilheit (e)
- Frequenz (f)
- Amplitude (c)
- Geschwindigkeit des Signals

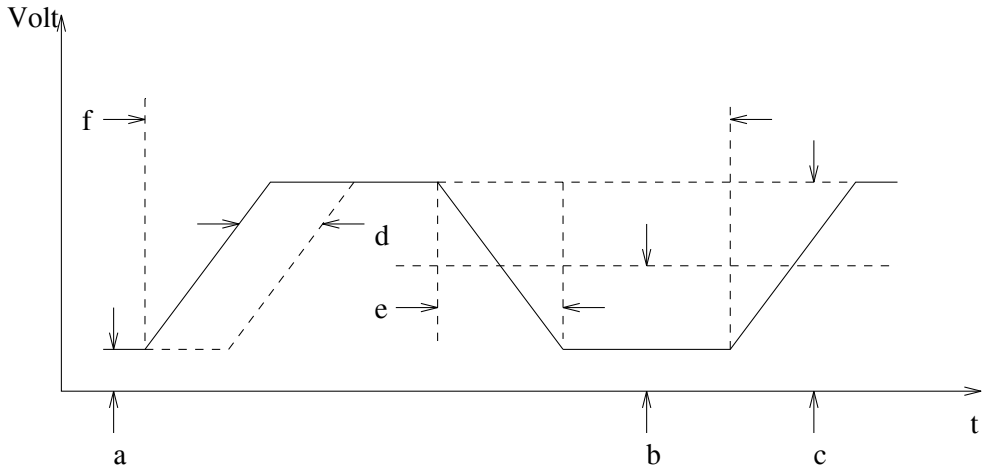


Abbildung 2.1: Meßgrößen der physikalischen Schicht

Die gemessenen Werte können zusammen mit statischen Angaben über das Netz benutzt werden, um zu überprüfen, ob sich die Leitung gemäß einer bestimmten Spezifikation verhält. Durch Analyse der Werte können weitere Aussagen getroffen werden, beispielsweise ob es einen Kabelbruch an einer Stelle im Kabel gibt. In Formel (2.1) gibt *dis* einen Näherungswert für die Entfernung eines möglichen Kabelbruches in Metern an. P ist die Ausbreitungsgeschwindigkeit des Signals im Kabel. TDR (Time Domain Reflectometer) gibt den Zählwert seit dem Beginn einer Übertragung bis zum Erkennen einer Kollision an. Der Wert wird mit der Bitrate des Mediums erhöht[1].

[1] Bei Ethernet sind das 10MHz ($10MHz * 1bit = 10Mbit/s$).

Der Faktor von $0,5$ ist nötig, da sich das Signal zum Kabelbruch hin ausbreiten muß und die Reflektion zurück zum Meßgerät kommen muß.

$$dis = \frac{1}{2} * P * TDR * 0.1 * 10^{-6} \tag{2.1}$$

Digitale Messungen der Schicht 1 beziehen sich meist auf die

- Bitfehlerhäufigkeit (engl. BER, Bit Error Rate)

$$BER = \frac{\text{fehlerhafte Bit}}{\text{Gesamtzahl übertragener Bit}} \tag{2.2}$$

- Übertragungsgeschwindigkeit (Bit/s)

Eine hohe bzw. für die gewählte Leitung hohe Bitfehlerhäufigkeit, kann obige analoge Messungen nach sich ziehen, da Bitfehler entstehen, wenn Meßwerte bei den analogen Messungen von der Spezifikation abweichen.

2.1.2 Schicht 2, Datensicherungsschicht

Hauptaufgabe der Sicherungsschicht ist, die Bitfehler der Schicht 1 zu erkennen und gegebenenfalls zu korrigieren. Dazu wird der Bitstrom der Schicht 1 als sequenziell übertragene Blöcke interpretiert, die einem bestimmten Format genügen müssen. Das Format wird in einer Protokollspezifikation festgelegt. Die 'Bitblöcke' sind somit als Schicht 2 Protokolldateneinheiten (PDUs) interpretierbar. In dieser Protokollschicht ist meßbar bzw. analysierbar:

- Anzahl fehlerhafter/fehlerfreier PDUs seit Beginn der Messung
- Anzahl fehlerhafter/fehlerfreier PDUs pro Zeiteinheit
- Verteilung der PDU-Typen (prozentualer oder ganzzahliger Anteil bestimmter PDUs an der Gesamtzahl aller PDUs)
- Prüfen der syntaktischen Korrektheit von PDUs
- Aufschlüsselung der PDUs gemäß ihrem PDU-Format (Anzeigen einzelner PDU-Felder, insbesondere der eingebetteten PDUs höherer Schichten)
- Angaben zur Längenverteilung der PDUs (vgl. [Sta93, S. 500ff])
- Angabe der Auslastung des Übertragungsmediums ('Utilization'), minimal, maximal, im Mittel, seit Beginn der Messung, pro Zeiteinheit
- Angaben zur Verkehrsbelastung einzelner Knoten
- Antwortzeiten (sofern das jeweilige Protokoll die Unterscheidung zwischen einer Aktion (hier Senden einer antwortauslösenden Nachricht) und Reaktion (hier Senden einer 'Antwort-Nachricht') erlaubt (beispielsweise beim 'Handshake' eines Verbindungsaufbaus).

- Testen einer Protokollinstanz (Stimulieren der zu prüfenden Instanz durch PDUs und Analysieren der 'Antwort-PDUs' bezüglich der Protokollspezifikation) (vgl. [GK94, S. 354ff])
- Testen von Komponenten, die in Netzen auf der Schicht 2 operieren (beispielsweise Test, ob eine 'Bridge' genau die PDUs passieren läßt, die gemäß ihrer Zieladresse passieren dürfen.)

2.1.3 Schicht 3, Vermittlungsschicht

Zusätzlich zu den Messungen der Schicht 2 sind hier Messungen nötig, die die typischen Funktionen der Schicht 3 überprüfen. Zu diesen Funktionen gehören vor allem die Adreßvergabe, die Wegwahl (Vermitteln) und eventuell das Aufbauen, Halten und Beenden von Verbindungen. Daraus ergeben sich folgende Meß- und Analyseaufgaben:

- Überprüfen der Bindung von physikalischen Adressen zu Schicht 3 Adressen (vgl. Abschnitt 5.11)
- Überprüfen der Wegwahl von Vermittlungsrechnern für PDUs
- Überprüfen der Funktionen zum Segmentieren und Reassemblieren von PDUs
- Zugriffskontrolle von fremden Netzen auf das eigene LAN und umgekehrt (für Sicherheitsüberprüfungen, 'Accounting', Gebührenberechnung usw.)

2.1.4 Schicht 4, Transportschicht

Zusätzlich zu den Messungen der Schicht 2 sind hier Messungen nötig, die die Aufgaben der Transportschicht überprüfen:

- Überprüfen von Flußkontrollmechanismen
- Überprüfen der Sequenzierung (kommt es zu Reihenfolgevertauschungen von PDUs ?)
- Überprüfen der Wahl eines Netzes bei mehreren alternativen Netzen (Ist das gewählte Netz in der Lage, die vom Benutzer gewünschte Dienstgüte zu erbringen ?)
- Angaben zu Verbindungen (Dauer, Anzahl übertragener Byte, Typ und zeitliche Anordnung der Nachrichten des 'Handshake')

2.1.5 Schicht 5–7, Kommunikationssteuerungs-, Darstellungs- und Anwendungsschicht

Die Aufgaben der anwendungsnahen Protokolle sind speziell auf die Bedürfnisse der Anwendungen zugeschnitten. Grundsätzlich sind hier ähnliche Analysen wie für die Schicht 2 möglich und eventuell Überprüfungen von Verbindungsauf- und Abbau, sowie Angaben zur Verbindung wie bei der Schicht 4.

2.1.6 Alternative Schichteneinteilung

Die oben an dem OSI-Schichtenmodell diskutierten Messungen können bei Bedarf auch zusammengefaßt werden. So werden in [F+87] beispielsweise Messungen zum 'Stations-Durchsatz' und zum 'Interface-Durchsatz' diskutiert. Beim 'Stations-Durchsatz' wird der Durchsatz an Benutzerdaten an der Schnittstelle zum Transportsystem gemessen. Analog bezieht sich der 'Interface-Durchsatz' auf die Schnittstelle einer Station zum Netz.

2.1.7 Schichtunabhängige Anforderungen an die Datenerfassung und Analyse

Unabhängig von der jeweiligen Protokollschicht muß ein universelles Meßwerkzeug

- eine genügend hohe Auflösung (Anzahl der 'Samples' pro Sekunde bzw. Anzahl verarbeitbarer Ereignisse pro Sekunde) für die zu erbringende Meßaufgabe haben,
- umfangreiche statistische Auswertungen der erhobenen Daten bieten,
- bei verteilter Datenerfassung die Ereignisse in eine globale Ordnung (Reihenfolge) bringen können.

2.2 Anforderungen an das Filtern der Daten

Die erfaßten Daten müssen in der Regel gefiltert werden, bevor sie weiterverarbeitet werden können. Unter *Filtern* wird hier die Selektion bestimmter Informationen aus den erfaßten Daten verstanden. Ziel des Filterns ist eine Reduktion des Datenvolumens. Die Reduktion der Daten soll nach Vorgaben des Benutzers erfolgen. Es ergeben sich mindestens folgende Anforderungen an das Filtern der Daten:

- Standardfilter für häufige Anwendungsfälle bzw. für häufig benutzte Protokolle (Protokolltyp, 'Portnummern', Sender- und Empfängeradressen)
- Benutzerdefinierbare Filter (Angabe einer Startposition, an der ein bestimmtes Bitmuster gefunden werden soll)
- Kombinationsmöglichkeiten der Filter über boolesche Funktionen (AND, OR, NOT)
- Verwalten von benutzerdefinierten (nicht-standard-) Filtern (Namensgebung, Speichern, Laden)

Zusätzlich zu den Filtern sollten universelle Meßgeräte auch sogenannte 'Trigger'-Mechanismen bieten. Ähnlich den Filtern arbeiten auch die 'Trigger' auf den Meßdaten. Der Unterschied zu den Filtern besteht darin, daß 'Trigger' keine Reduktion der Daten vornehmen, sondern abhängig von den erhobenen Daten benutzerdefinierte

Aktionen auslösen. Durch 'Trigger' läßt sich das Verhalten des Meßgerätes programmieren. Beispielsweise könnte ein 'Trigger' das Erfassen von Meßdaten ab einer bestimmten Uhrzeit veranlassen. Ein weiterer 'Trigger' beendet die Meßdatenerfassung beispielsweise nach dem Eintreffen einer bestimmten Anzahl an 'Ethernet-Frames'.

2.3 Anforderungen an das Bewerten der Daten

'Trigger' lassen sich außerdem für das Bewerten der Daten durch den Benutzer einsetzen. Unabhängig vom Benutzer können Bewertungen durch ein Expertensystem erfolgen. Die Bewertungsmöglichkeiten ergeben sich durch:

- Einsetzen der 'Trigger'-Mechanismen für benutzerdefinierte Fehler- und Ereignismeldungen und einer
- Schnittstelle zu Expertensystemen.

Ein 'Trigger' könnte beispielsweise das Eintreffen von ICMP-Nachrichten[2] des Typs 'ECHO-Request' erkennen und die Meldung 'Netzmanagement von Station XYZ' ausgeben. Dabei wird XYZ durch die jeweilige Senderadresse ersetzt.

Die Programmierung derartiger 'Trigger' setzt ein bestimmtes Maß an Expertenwissen voraus. Eine Schnittstelle zu einem Expertensystem könnte eventuell hilfreich sein, falls das Meßwerkzeug zur Fehlersuche in einem heterogenen Netz eingesetzt wird, das auch für Experten aufgrund der Geräte- und Protokollvielfalt unübersichtlich ist.

2.4 Anforderungen an das Anzeigen der Daten

Die Visualisierung der Meßdaten und der Analyseergebnisse ist wichtig für eine Bewertung der Messung durch die Benutzer. Grundsätzlich müssen Meßwerte

- tabellarisch und
- graphisch

darstellbar sein. Die Wahl des jeweiligen Anzeigeformates sollte immer dem Benutzer überlassen werden. Typische Diagrammformen für Meßwerte sind

- Liniendiagramme,
- Histogramme,
- Kiviat-Diagramme und
- Gantt-Diagramme.

Für eine ausführliche Darstellung vgl. [FSZ83, S. 174ff]. Darüber hinaus sind für die Visualisierung verschiedener Aspekte der dynamischen Protokollverarbeitung in einem Kommunikationssystem weitere graphische Darstellungen denkbar:

[2] Das 'Internet Control Message Protocol' wird in Abschnitt 5.3.2.5 ausführlich beschrieben.

- Balkendiagramme ('Speedbars'), beispielsweise zum Anzeigen der gegenwärtigen Last.
- Zeitfolgediagramme, die automatisch für zu analysierende Verbindungen erstellt werden.
- Diagramme mit Darstellungen der Netztopologie (Endgeräte, Segmente, Brücken, Vermittlungsrechner usw.).

2.5 Anforderungen an das Verwalten der Daten

Die Meßdaten müssen in der Regel für eine spätere Weiterverarbeitung verwaltet werden. Anforderungen an die Verwaltung der Meßdaten sind:

- Exportfunktionen für die Daten
- Schnittstelle zu Datenbanken
- Schnittstelle zu 'Remote Monitoring Management Information Base' (RMON-MIB) über das 'Simple Network Management Protocol' (SNMP)
- Verschlüsselung sensibler Daten

Besonders wichtig ist hier die Exportfunktion. Alle Meß- und Analysedaten sollten in einem Standardformat (z.B. US-ASCII, ISO-Latin1) exportierbar sein. Dazu werden die erfaßten Daten gegebenenfalls von einer internen Darstellung in ein Standardformat gewandelt und anschließend in eine Datei geschrieben. Die so erstellte Datei kann als Grundlage für alle weiteren oben aufgeführten Funktionen in externen Werkzeugen dienen. Auch wenn das Meßwerkzeug selbst nicht in der Lage ist, die exportierten Daten weiterzuverarbeiten, können viele externe Werkzeuge die Daten aus der Datei lesen.

Der Export der Daten in eine Datei ist ein allgemeines Konzept, um Unzulänglichkeiten des Meßwerkzeuges auf (fast) allen Ebenen ausgleichen zu können. Es kann somit auch als Grundlage für die Erweiterung des Werkzeuges genutzt werden. Reichen die eingebauten Analyse- und Anzeigefunktionen nicht aus, so kann durch den Export mit anderen Werkzeugen auf die Daten zugegriffen werden.

Im Falle von Meßgeräten für Verteilte Systeme kommen weitere Anforderungen dazu:

- Zusammenführen von dezentral erhobenen Daten in eine globale Ereignisreihenfolge
- Versenden von dezentral erhobenen Daten über ein Netz

Eine weiterführende Diskussion der Funktionen für Meßwerkzeuge in Verteilten Systemen findet man in [Jai91, S. 101ff], sowie in [MSS93] und [K$^+$95].

2.6 Anforderungen der Benutzer

Die Anforderungen der Benutzer sind üblicherweise[3]:

- leichte und komfortable Bedienbarkeit, insbesondere
- Wahl der Schnittstelle aus graphischer Oberfläche oder Kommandozeile
- Konformität zu Standardschnittstellen ('Windows', 'CUA' usw.)
- Wahl zwischen Bedienmodi 'Anfänger', 'Fortgeschrittener' und 'Experte'
- Eingabehilfen, z.B. vorkonfigurierte Meßaufgaben, 'Meßtemplates'
- Fehlertoleranz bei Fehlbedienung
- Hilfsfunktionen (kontextabhängige Hilfe, 'On-Line-Hilfe', Hypertextdokumentation, Referenzhandbücher)
- freie Kombinationsmöglichkeiten der vorhandenen Funktionen (Orthogonalität)
- Erweiterbarkeit, insbesondere durch:
  - Programmierschnittstelle
  - Anpassen der 'On-Line-Dokumentation'
  - Anpassen an neue Protokolle
  - Anpassen an neue Netze (u.a. auch an physikalische Übertragungsarten)

Es sind insbesondere die oben genannten Benutzeranforderungen, die bei der Implementation einen streng modularen Aufbau erforderlich machen. Die Wahl einer bestimmten Schnittstelle kann dann leicht durch alternative 'Schnittstellenmodule' ermöglicht werden. Die Anpassung an neue Protokolle geschieht durch 'Protokollmodule', die im Sinne der Orthogonalität die vorhandene Funktionalität zur Analyse auf die neuen Protokolle ausdehnen.

2.7 Anforderungen an ein paralleles Meßwerkzeug für Hochgeschwindigkeitsnetze

In diesem Abschnitt soll verdeutlicht werden, daß heutige und zukünftige Anforderungen an Meßwerkzeuge für Hochgeschwindigkeitsnetze parallele Realisierungen implizieren.

Die bisher in diesem Kapitel aufgestellten Anforderungen an ein allgemeines Meßwerkzeug für Kommunikationssysteme sind als Ideen zu verstehen, die bei einer Realisierung berücksichtigt werden können.

[3] Viele der genannten Benutzeranforderungen gelten für alle nicht-trivialen Realisierungen, die eine Mensch-Maschine-Schnittstelle haben.

Diese Liste ist zwangsläufig unvollständig, da sich die Anforderungen der Benutzer jederzeit ändern können. Es werden daher in dieser Diplomarbeit einige rudimentäre Funktionen realisiert, die die Anwendungsgebiete des parallelen Meßwerkzeuges verdeutlichen (vgl. Abschnitt 2.7.4).

Wichtiger als die Wahl zu realisierender Funktionen sind im Kontext von Hochgeschwindigkeitsnetzen jedoch Anforderungen bezüglich der Erweiterbarkeit (vgl. Abschnitt 2.7.1), der Leistungsfähigkeit (vgl. Abschnitt 2.7.2) und der Skalierbarkeit (vgl. Abschnitt 2.7.3).

2.7.1 Anforderungen an die Erweiterbarkeit

Ein Werkzeug sollte grundsätzlich erweiterbar sein, um neuen Benutzeranforderungen gerecht zu werden.

Es gibt genau zwei Möglichkeiten, ein Werkzeug um neue Funktionalität zu erweitern. Einerseits kann man externe Werkzeuge benutzen, um die exportierten Daten weiterzuverarbeiten. Andererseits kann man neue Funktionalität direkt in das Werkzeug einbauen.

Daraus ergeben sich die folgenden Minimalanforderungen:

- Die analysierten Daten müssen durch Exportfunktionen in eine ACSII-Datei geschrieben werden.
- Das Format der exportierten Daten sollte einer maschinell leicht erkennbaren Syntax genügen, um den externen Werkzeugen einen effizienten Zugriff auf die Daten zu ermöglichen.
- Die Grundfunktionalität muß modular programmiert werden, um spätere Erweiterungen zu vereinfachen.
- Die Schnittstellen dieser Module sind zu dokumentieren.

2.7.2 Anforderungen an die Leistungsfähigkeit

Hochgeschwindigkeitsnetze (HSN) zeichnen sich insbesondere durch hohe Übertragungsgeschwindigkeiten aus. Ein Meßwerkzeug für Hochgeschwindigkeitsnetze muß grundsätzlich in der Lage sein, alle Dateneinheiten ohne Verluste auszuwerten. Dies ist ein wesentlicher Unterschied zwischen einem Endsystem und einem dedizierten Meßwerkzeug (vgl. Abschnitt 3.3.2). Endsysteme empfangen nur die Dateneinheiten, die fr sie bestimmt sind.

Diese gegenüber einem Endsystem vielfach höhere Anforderung an die Leistungsfähigkeit ist der Hauptgrund für die Entscheidung, ein solches Meßwerkzeug auf einer parallelen Rechnerarchitektur zu realisieren.

Ein weiterer Grund für die Wahl einer parallelen Rechnerarchitektur ist die Einschätzung, daß das Wachstum in den Übertragungsgeschwindigkeiten von HSN[4] in Zukunft

[4]Zur Zeit betrachten wir Netze mit einer Brutto-Übertragungsgeschwindigkeit von mindestens 100 Mbit pro Sekunde als Hochgeschwindigkeitsnetze. Zur Zeit sind Prototypen von Netzen mit

deutlich größer sein wird, als der Zuwachs der Rechenleistung neuer Prozessoren. Da es schon heute kaum noch möglich ist, mit einem Einprozessorsystem den Durchsatz der Netze dem Benutzer in einem Endsystem verfügbar zu machen (vgl. [Zit91a, S. 6f]), ist es höchst unwahrscheinlich, daß Meßwerkzeuge, basierend auf Einprozessor-Architekturen, mit den Übertragungsgeschwindigkeiten in Zukunft mithalten können.

2.7.3 Anforderungen an die Skalierbarkeit

Die Einschätzung der enormen Übertragungsgeschwindigkeiten zukünftiger Netze erfordert eine skalierbare Architektur, die an die steigenden Übertragungsgeschwindigkeiten der Netze angepaßt werden kann. Die Forderung nach Skalierbarkeit schlägt sich vor allem in der Wahl eines geeigneten Paradigmas zur parallelen Protokollverarbeitung nieder. Daher wird dieser Aspekt erst in den Abschnitten 3.3.3 bis 4.2 diskutiert.

Unmittelbar einsichtig ist, daß nicht mit jedem neuen HSN ein neuer Monitor konzipiert werden sollte. Vielmehr muß die Skalierung des Monitors mit der Skalierung der Netze mithalten.

2.7.4 Zusammenfassung der Anforderungen

Die nun folgende Liste der Anforderungen an ein paralleles Meßwerkzeug muß unter folgenden Restriktionen realisiert werden:

- Das Meßwerkzeug soll im Rahmen einer Diplomarbeit konzipiert, realisiert und bewertet werden. Diese zeitliche Restriktion[5] hat im wesentlichen Auswirkungen auf den realisierbaren Funktionsumfang.
- Es muß veraltete Hardware benutzt werden, da Neubeschaffungen für diese Diplomarbeit nicht möglich sind. Insbesondere gibt es keinen Zugang zu einem Hochgeschwindigkeitsnetz. Es muß deshalb mit einem Ethernet-Netz gearbeitet werden. Die Restriktionen durch die Hardware werden im Abschnitt 4.1 diskutiert.
- Die Sicherheit des Netzes und die der daran angeschlossenen Rechner dürfen durch das Meßwerkzeug nicht beeinträchtigt werden. Das Meßwerkzeug soll ausschließlich passiv das Verkehrsaufkommen im Netz beobachten. Dies hat ebenfalls Auswirkungen auf den Funktionsumfang der Realisierung. Es wird insbesondere darauf verzichtet:
  - Empfangene Benutzerdaten in Dateien zu speichern,
  - Zugangsberechtigungen und dazugehörige Paßwörter zu speichern oder
  - durch Versenden von Dateneinheiten seitens des Meßwerkzeuges das zu untersuchende Netz zu beeinflussen.

Übertragungsgeschwindigkeiten in der Größenordnung von einem Gigabit pro Sekunde realisiert und die Forschung geht in Richtung 'Terabit-Netze'.

[5] Die zur Verfügung stehende Zeit beträgt ein Jahr.

Die Anforderungen an ein paralleles Meßwerkzeug sind unter Berücksichtigung der Restriktionen:

- Realisierung auf Grundlage eines noch zu wählenden Paradigmas für die parallele Protokollanalyse (vgl. Abschnitt 3.3.3).
- Erfassen von Meßwerten an noch zu definierenden Schnittstellen (vgl. Abschnitt 4.4.2.1).
- Spezifizieren dieser Schnittstellen mit Hilfe von 'Filtern' (vgl. Abschnitt 2.2).
- Protokollieren eines Teils der TCP/IP-Protokollfamilie.
- Auswahl der zu analysierenden Meßwerte und Protokolle durch den Benutzer.
- Auswahl der zu erfassenden Meßwerte über 'Dialoge' einer graphischen Benutzerschnittstelle.
- Anzeigen einer 'Standardauswertung' während des Betriebs über eine graphische Benutzerschnittstelle.
- Exportieren der Meßergebnisse in eine Datei für Auswertungen mit externen Werkzeugen.
- Die Erweiterbarkeit muß durch einen modularen Aufbau gewährleistet sein.

Kapitel 3

Abbilden der 'Monitor-Prozesse' auf die Hardware

Nachdem im vorherigen Abschnitt die Anforderungen an ein Meßwerkzeug aufgestellt wurden (vgl. Abschnitt 2.7) und die grundsätzliche Notwendigkeit für eine parallele Realisierung motiviert wurde, kann in diesem Kapitel gezielt auf die Parallelverarbeitung eingegangen werden.

Es sollen grundsätzliche Möglichkeiten für die Realisierung des Werkzeuges diskutiert werden. Nach der Festlegung der funktionalen Anforderungen muß vor der Realisierung überprüft werden, wie diese Funktionen strukturiert werden können. Ziel einer Strukturierung ist die Festlegung, welche Funktionen zu Prozessen zusammengefaßt werden. Die Prozesse wiederum sind die elementaren Einheiten, die auf die verfügbare Hardware abgebildet werden. In Abschnitt 3.1 wird ein Überblick über theoretische Aspekte dieser *Zuordnungsprobleme* gegeben.

Ein wichtiges Kriterium für die Zuordnung der Funktionen ist die Hardware (HW), da sich aus deren Eigenschaften eine Vielzahl von Randbedingungen (Restriktionen) für das Plazieren von Prozessen ergeben. Eine Darstellung der zur Verfügung stehenden HW und der Entwicklungswerkzeuge ist in Abschnitt 3.2.

In Abschnitt 3.3 werden Konzepte zur parallelen Protokollverarbeitung unter den Aspekten der zu implementierenden Funktionen, den Einschränkungen durch die HW, der Erweiterbarkeit und der Leistungsfähigkeit diskutiert. Die Entscheidung für eines dieser Konzepte legt wesentliche Aspekte für die Realisierung des Werkzeuges fest.

In Kapitel 4 kann auf Grundlage der hier vorgestellten Möglichkeiten und Restriktionen der Entwurf eines Prototypen diskutiert werden.

3.1 Das Zuordnungsproblem der Parallelverarbeitung

In diesem Abschnitt soll das Problem der Zuordnung von Programmen und Prozessen zu den verfügbaren Prozessoren eines Parallelrechners dargestellt werden. Das Problem der Zuordnung hat drei wesentliche Aspekte. Einerseits ist eine Zuordnung abhängig von der Hardware (vgl. Abschnitt 3.2), andererseits von dem *Zuordnungsziel*

(vgl. Abschnitt 3.1.3). Außerdem gibt es bei der Zuordnung einen zeitlichen Aspekt. Die Entscheidung, welche Instanzen bei der Programmentwicklung und Programmausführung die Zuordnung vornehmen, hat Einfluß auf das dynamische Verhalten eines parallelen Programms (vgl. Abschnitt 3.1.4).

3.1.1 Zuordnung von Algorithmen zu Prozessoren

Ein *Programm* realisiert einen oder mehrere Algorithmen. Das Programm muß zur Berechnung auf mindestens einem *Prozessor* ausgeführt werden. Zu den Problemstellungen der sequentiellen Programmierung, wie beispielsweise die *prozedurale Dekomposition* der Algorithmen in überschaubare kleinere Teilalgorithmen, kommen bei der parallelen Programmierung die beiden folgenden Problemkomplexe hinzu:

- Aufspüren der *inhärenten Parallelität* eines Algorithmus
- Abbilden der parallelen Programme auf die Prozessoren

Die inhärente Parallelität eines Algorithmus besteht aus denjenigen parallel ausführbaren Teilalgorithmen, die nur aufgrund der bisher dominierenden sequentiellen Programmierung *nacheinander* ausgeführt wurden. Diese inhärente Parallelität muß "wiederentdeckt" werden (vgl. Abschnitt 3.3).

Realisiert man die inhärente Parallelität eines Algorithmus als nebenläufige *Prozesse*, erhält man parallele Programme, die aus einer Menge kooperierender[1] Prozesse bestehen. Der zweite Problemkomplex ist dann das Abbilden (Zuordnen, engl. "mapping") der parallelen Programme bzw. der Prozesse auf die Prozessoren des Parallelrechners.

3.1.2 Formale Präzisierung der Zuordnung

Einige Begriffe der Zuordnungsproblematik lassen sich formal darstellen:

$$\begin{aligned} P \quad & \text{Menge der Prozessoren } (|P| = n); \\ A := \{A_1, A_2, \cdots A_{|A|}\} \quad & \text{Last, bestehend aus einer} \\ & \text{Menge paralleler Programme;} \\ T_i \quad & \text{Menge der Prozesse des Programms } A_i \\ & (i = 1, \cdots, |A|). \end{aligned} \tag{3.1}$$

Als *Programmzuordnung* bezeichnet man die Abbildung von der Menge der Programme (A) auf die Potenzmenge der Prozessoren ($\wp(P)$), da durchaus zwei oder mehr Programme einem Prozessor zugeordnet werden können.

$$\text{Programmzuordnung:} \quad \varphi : A \to \wp(P) \tag{3.2}$$

Will man eine disjunkte Abbildung der Programme auf die Prozessoren, so daß ein Prozessor einem Programm exklusiv zur Verfügung steht, kann man eine *disjunkte*

[1] Die Kooperation der Prozesse besteht darin, daß sich zwei oder mehr Prozesse *synchronisieren* müssen, um beispielsweise Daten auszutauschen.

Programmzuordnung durch folgende Abbildung erreichen:

$$\text{disjunkte Programmzuordnung:} \quad \varphi : A \to \wp(P), \quad \forall i \neq k : \quad \wp(A_i) \cap \wp(A_k) = \emptyset \tag{3.3}$$

Die Programmzuordnung ist demnach entweder disjunkt oder *überlappend*.

Die Prozessoren sind durch ihre Verbindungen untereinander zu einer bestimmten *Topologie* angeordnet, die sich graphentheoretisch als Menge von Knoten (Prozessoren) und Menge von gerichteten oder ungerichteten Kanten (Verbindungen) darstellen lassen. Eine Zuordnung eines Programms A_i heißt *zusammenhängend*, wenn der durch die Zuordnung entstehende Untergraph (Menge der belegten Knoten und deren Kanten) zusammenhängend ist. Eine nicht zusammenhängende Zuordnung bezeichnet man als *gestreut*.

Bei einer disjunkten Partitionierung kann man darüber hinaus von der Zuordnung zu bestimmten Prozessoren abstrahieren und ausschließlich festlegen, wieviele Prozessoren ein Programm belegen soll. Man bezeichnet diese Abbildung von der Menge der Programme auf die Prozessorzahlen ($\{1, \cdots, n\}$) als *quantitative Partitionierung*:

$$\text{quantitative Partitionierung:} \quad \chi : A \to \{1, \cdots, n\} \text{ mit } \sum_{i=1}^{|A|} \chi(A_i) \leq n \tag{3.4}$$

Dabei können an alle Programme (A) nur so viele Prozessoren vergeben werden, wie tatsächlich vorhanden sind (hier n). Dies wird durch den Term mit der Summe über alle Programme in Formel 3.4 sichergestellt.

Bei der *Prozeßzuordnung* werden nicht mehr die Programme den Prozessoren zugeordnet, sondern die Menge aller Prozesse aller Programme (T) wird auf die Prozessoren abgebildet:

$$\text{Prozeßzuordnung:} \quad \pi : T \to P \tag{3.5}$$

Sind mindestens so viele Prozessoren vorhanden wie Prozesse ($|T| \leq n$), kann man jedem Prozeß einen Prozessor zuteilen. Eine solche Abbildung heiß *injektiv*:

$$\forall i, k \in \{1, \cdots, |T|\}, i \neq k : \qquad \pi(i) \neq \pi(k) \tag{3.6}$$

Werden zwei oder mehr Prozesse auf einen Prozessor abgebildet, heißt die Zuordnung *kontraktiv*. Will man stets eine injektive Zuordnung von Prozessen zu Prozessoren erreichen, muß man bei $|T| \geq n$ Prozessen eine *unvollständige* Zuordnung machen, bei der eine Teilmenge der Prozesse nicht zugeordnet wird. Bei Kontraktion oder wenn gilt $|T| \leq n$, kann man eine *vollständige* Zuordnung erreichen.

Aus der Sicht der Prozeßverwaltung eines Parallelrechners kann sich eine Zuordnung mit der Zeit ändern. Das hat hauptsächlich zwei Gründe:

1. bereits aktive Prozesse terminieren und geben einen Prozessor frei
2. neue Programme sollen abgearbeitet werden bzw. neue Prozesse werden dynamisch generiert und müssen auf (freie) Prozessoren abgebildet werden.

Die Gesamtmenge der zu einem Zeitpunkt t zu verwaltenen Prozesse (T_t) setzt sich zusammen aus den *aktiven* Prozessen, die bereits zum Zeitpunkt t auf Prozessoren

plaziert sind und den *wartenden* Prozessen (T_t^w), die bis zum Zeitpunkt t neu hinzugekommen sind. Bezeichnet man die Menge der zum Zeitpunkt t nicht belegten Prozessoren mit P_t^f, so kann man den dynamischen Verlauf der Belegung als *konservativ* oder als *aggressiv* bezeichnen:

$$\begin{aligned} \pi_{konservativ}: &\quad T_t^w \to P_t^f \\ \pi_{aggressiv}: &\quad T_t \to P \end{aligned} \tag{3.7}$$

Der Unterschied besteht darin, daß bei der aggressiven Zuordnung Prozesse von den Prozessoren verdrängt werden können. Diese können entweder einem anderen Prozessor zugeordnet werden (Migration) oder eine bestimmte Zeit lang warten, bis sie wieder zugeordnet werden. Bei der konservativen Zuordnung werden nur solche Prozessoren belegt, die ihre vorherigen Aufträge ausgeführt haben.

Diese Definitionen sind eine Zusammenfassung der Darstellung in [Hei94, S. 82ff]. In Abbildung 3.1 ist eine Klassifikation für die Zuordnung abgebildet.

Abbildung 3.1: Klassifikation der Zuordnung (vgl. [Hei94, S. 87])

3.1.3 Ziele bei der Zuordnung

Eine Zuordnung von Prozessen[2] zu Prozessoren ist nicht willkürlich, sondern zielgerichtet. Man möchte eine *optimale* Zuordnung entsprechend einer oder mehrerer Zielfunktionen erreichen. Diese Zuordnung muß im allgemeinen Fall noch unter bestimmten Randbedingungen, den *Restriktionen*, erfolgen.

Zu optimierende Zielfunktionen können u.a. sein (vgl. [Hei94, S. 89]):

- Durchsatz
- gleichmäßige Auslastung (der Prozessoren)
- Antwortzeiten

Restriktionen können entweder *qualitativ* oder *quantitativ* sein. Beispiele für qualitative und quantitative Restriktionen sind:

- Prozeß j darf ausschließlich auf Prozessor x plaziert werden (qualitativ).
- Die Speicheranforderungen eines Prozesses j darf auf Prozessor x zu keinem Zeitpunkt größer als z MByte sein (quantitativ).
- Die mittlere Antwortzeit eines Prozesses des Typs j soll kleiner als m Millisekunden sein (quantitativ).

Kombiniert man die Optimierung mit Restriktionen, so erhält man eine *bedingte Optimierung*. Eine Optimierung ohne Restriktionen ist *unbedingt*. Abbildung 3.2 zeigt eine mögliche Klassifizierung der Zuordnungsziele.

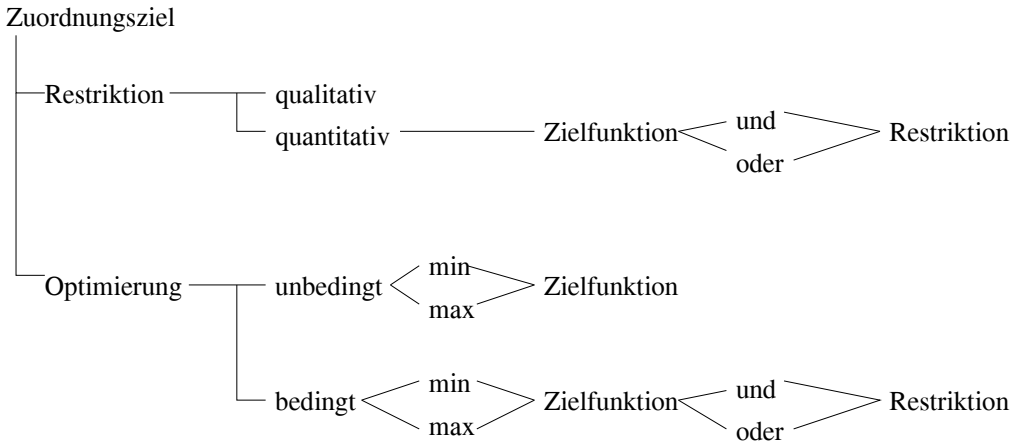


Abbildung 3.2: Klassifikation der Zuordnungsziele (vgl. [Hei94, S. 90])

[2]Es wird nur noch auf den allgemeinen Fall der Zuordnung von Prozessen zu Prozessoren eingegangen, da Programme als Menge von kooperierenden Prozessen verstanden werden.

3.1.4 Zeitpunkt der Zuordnung

Der Zeitpunkt der Zuordnung von Prozessen zu Prozessoren ist nicht fest vorgegeben, sondern fällt in eine der folgenden Kategorien.

3.1.4.1 Manuelle Zuordnung

Bei der manuellen Zuordnung entscheidet der Programmierer die Zuordnung der Prozesse eines Programms zu den Prozessoren. Diese Methode ist:

- 'hardwarespezifisch'. Eine manuelle Zuordnung muß für jeden Parallelrechner, auf dem das Programm eingesetzt werden soll, erneut gemacht werden.
- statisch. Eine einmal gemachte Zuordnung ändert sich nicht über der Zeit.
- zeitaufwendig und arbeitsintensiv. Mit steigender Anzahl an Prozessoren wird der Aufwand für den Programmierer für die Zuordnung größer.
- fehleranfällig. Manuelle Zuordnungen sind fehleranfällig, insbesondere weil der Programmierer die Einhaltung der Restriktionen (s.o.) selber überprüfen muß.
- ungeeignet für bestimmte Rechnerarchitekturen. Nur für eine "überschaubare" Menge von Prozessen und Prozessoren[3] ist eine manuelle Zuordnung vorstellbar (z.B. Anzahl der Prozessoren $\ll 100$). Bei einigen Rechnerarchitekturen ist sie sogar unmöglich (beispielsweise bei Datenflußrechnern).
- geeignet für irregulär vermaschte Prozessoren. Besonders bei hochspezialisierten Topologien, bestehend aus einer bestimmten Anzahl von Prozessoren mit unterschiedlichen Fähigkeiten, ist dies oft die einzig mögliche Methode für die Zuordnung.

3.1.4.2 Zuordnung zur Übersetzungszeit

Für eine große Anzahl von Prozessoren ist eine manuelle Zuordnung zu kompliziert und zu zeitaufwendig. Geeignete Werkzeuge können dem Programmierer viel Arbeit abnehmen. Beispielsweise können Übersetzer für (explizit parallele) Programmiersprachen[4] Sprachkonstrukte für parallele Anweisungsfolgen als Grundlage für eine Zuordnung zur Übersetzungszeit nehmen. Teilweise lassen sich auch automatisch inhärente Parallelitäten (vgl. Abschnitt 3.1.1) erkennen und dementsprechend als eigenständige Prozesse realisieren. Ein Beispiel dafür wäre die Parallelisierung einer (sequentiellen) Wiederholungsanweisung. Diese Methode ist:

- 'hardwareunabhängig'. Der entsprechende Übersetzer muß natürlich für die Architektur verfügbar sein.

[3] Eine genaue Anzahl von Prozessen und Prozessoren kann man freilich nicht angeben. Je nach Erfahrung, zur Verfügung stehender Werkzeuge und Komplexität der Verknüpfungen einzelner Prozesse oder Prozessoren kann eine Grenze für das "Menschenmögliche" angegeben werden.

[4] Beispielsweise OCCAM, Concurrent Prolog, Linda u.a. (vgl. [Pen92], [Car88]).

- kaum fehlerträchtig. Übersetzer werden häufig benutzt. Fehler durch falsche Zuordnungsalgorithmen der Übersetzer werden meist bei Folgeversionen des Übersetzers behoben.
- für viele Prozesse und Prozessoren geeignet. Dies setzt eine gewisse Skalierbarkeit der Zuordnungsalgorithmen des Übersetzers voraus. Vorteilhaft bei wachsender Prozessorzahl ist auch eine reguläre Verknüpfung der Prozessoren (beispielsweise konstante Anzahl von 'Nachbarn' für alle Prozessoren).
- auch für ungeübte (Parallel-) Programmierer geeignet.

3.1.4.3 Zuordnung zur Ladezeit

Das Betriebssystem des Parallelrechners (sofern vorhanden) kann die Zuordnung selber unmittelbar vor der Ausführung machen. Diese Methode ist:

- gut geeignet, um die Last möglichst gleichmäßig auf die Prozessoren zu verteilen, da sie abhängig gemacht werden kann von der aktuellen Last zur Ladezeit.
- relativ fehlertolerant. Ist ein Prozessor zur Ladezeit als defekt erkannt worden, können Prozesse auf andere Prozessoren geladen werden.

3.1.4.4 Zuordnung zur Laufzeit

Bei der Zuordnung zur Laufzeit wird die oben beschriebene aggressive Zuordnung benutzt (vgl. Abschnitt 3.1.2). Diese Zuordnung ist daher:

- besonders gut an die sich ändernde Last anpaßbar. Durch Messung kann das Betriebssystem (theoretisch) die Auslastung der Prozessoren periodisch bestimmen, um möglichst gleichmäßig alle Prozessoren zu belasten (Lastausgleich, engl. 'load balancing', vgl. [Hei94, S. 279ff]).
- besonders anspruchsvoll. Die Algorithmen zur Messung und Bewertung von verteilten Lasten müssen Teil des Betriebssystems sein. Außerdem muß es Algorithmen zur Bewertung der *Migrationskosten* geben, damit der Gewinn durch eine Prozeßmigration zum Lastausgleich nicht durch die Kommunikationskosten (Zeit) wieder ausgeglichen wird.
- besonders rechenaufwendig. Die oben genannten Algorithmen für die dynamische Verteilung sind in der Regel nicht nur anspruchsvoll, sondern auch rechenintensiv (meist NP-vollständig!). Der Aufwand steigt also exponentiell mit der Anzahl der Prozessoren und den darauf zu verteilenden Prozessen, falls man optimale Lösungen sucht.

Ergänzend zu den hier gemachten Angaben, vgl. [Hei94, S. 95ff] zu "Basisalgorithmen" für Zuordnungen nach bestimmten Optimierungskriterien.

3.2 Verfügbare Hardware und Entwicklungsumgebung

In diesem Abschnitt werden die eingesetzte Hardware (HW) und die verschiedenen Entwicklungswerkzeuge beschrieben, die für die Realisierung des Meßwerkzeuges benutzt werden. Aus der HW (vgl. Abschnitt 3.2.1.7) ergeben sich einige Restriktionen bezüglich der Zuordnung von Prozessen zu Prozessoren, die weiter unten (Abschnitt 4.1) dargestellt werden.

Die zur Verfügung stehenden Entwicklungswerkzeuge (vgl. Abschnitt 3.2.2), insbesondere der einzusetzende Compiler, haben starken Einfluß auf die Umsetzung der Algorithmen des Meßwerkzeuges. Der Compiler muß die vorhandene HW möglichst optimal unterstützen.

3.2.1 Transputer und 'Transputer-Boards' für PCs

Die Beschreibung der HW läßt sich in drei Bereiche unterteilen. Wesentlich sind die Leistungsmerkmale der parallel arbeitenden Prozessoren (vgl. Abschnitt 3.2.1.1). Diese sogenannten 'Transputer' werden zusammen mit Speicher und weiteren spezialisierten Funktionseinheiten als 'Transputer-Boards' (TBs) bereitgestellt (vgl. Abschnitt 3.2.1.5). Diese TBs verfügen über Busschnittstellen für verschiedene 'Hostrechner'. In unserem Fall handelt es sich hierbei um 'Personal Computer' (PCs). Der Zugriff der PCs auf die Transputer wird in Abschnitt 3.2.1.6 dargestellt.

Zusammenfassend wird in Abschnitt 3.2.1.7 die Gesamtarchitektur der eingesetzten HW dargestellt. Diese Architektur ist die Grundlage für die Realisierung des Meßwerkzeuges.

3.2.1.1 Transputer

Bei den Transputern handelt es sich um eine Mikroprozessorfamilie der Firma INMOS Ltd. [Inm88],[Inm90]. Die Unterschiede zu anderen Mikroprozessoren fallen schon bei der Betrachtung des allgemeinen Blockschaltbildes für alle Transputer auf.

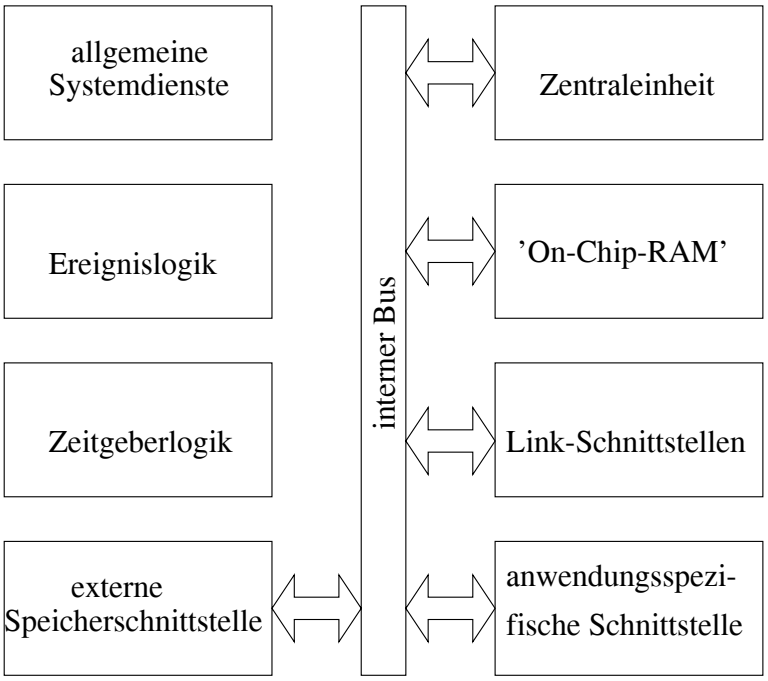


Abbildung 3.3: Allgemeines Blockschaltbild für alle Transputer (vgl. [Ebe93, Seite 3])

Transputer verfügen über:

- Internen Bus
- Zentraleinheit (Prozessor, CPU)
- 'On-Chip-Ram'(bereits vorhandenes RAM, transputerintern)
- 'Link-Schnittstellen'
- Zeitgeberlogik
- Ereignislogik (Interrupt-Schnittstelle)
- Systemdienste (Taktgenerierung, Speicherkonfigurationen, etc.)
- anwendungsspezifische Schnittstelle
  - Externe-Speicherschnittstelle
  - Gleitkomma-Einheit (nur bei T8xx-Typen)
  - Festplattenschnittstelle (nur bei M212)

3.2.1.2 Zentraleinheit

Die CPU des Transputers ist speziell für die Ausführung quasiparalleler Prozesse ausgelegt. Um schnelle Kontextwechsel zwischen den Prozessen zu erreichen, wurde der Mechanismus für die Prozeßumschaltung in die Zentraleinheit integriert. Ein Transputer benötigt somit keinen eigenen Software-'Scheduler' für die Prozeßverwaltung. Ein Kontextwechsel kann in ca. 1μs auf einem Transputer vorgenommen werden.

Es werden zwei Prioritätsstufen unterschieden. Prozesse hoher Priorität unterbrechen Prozesse niedriger Priorität sofort. Die Zustände der Prozesse auf einem Transputer und deren Übergänge sind in Abbildung 3.4 dargestellt.

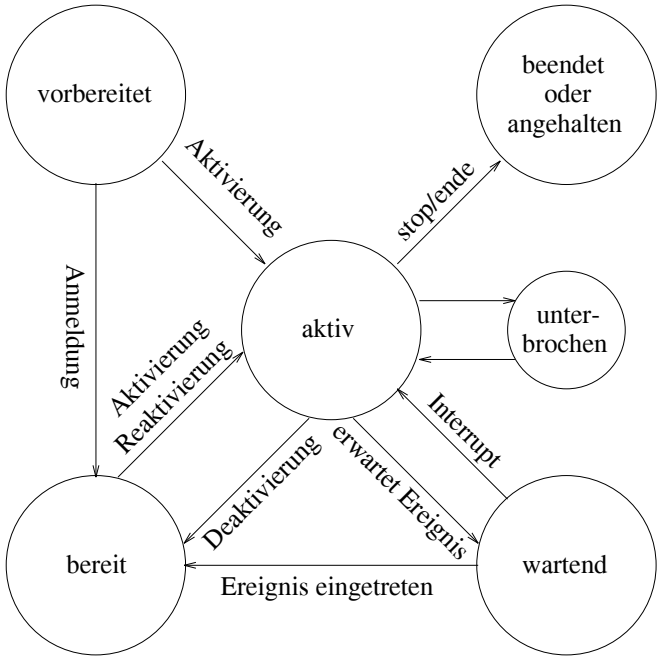


Abbildung 3.4: Zustandsdiagramm quasiparalleler Prozesse auf einem Transputer (vgl. [Ebe93, Seite 389])

Ein Prozeß ist *'vorbereitet'*, wenn sein Maschinencode im Speicher des Transputers steht. Der Vaterprozeß kann ihn beim 'Scheduler' des Transputers mit den Maschinenbefehlen 'startp' oder 'runp' anmelden. Der 'Scheduler' hängt diesen neuen Prozeß an die Prozeßliste. Der Prozeß wartet nun solange im Zustand *'bereit'*, bis ihn der 'Scheduler' bei der Abarbeitung der Prozeßliste aktiviert. Dann geht der Prozeß von bereit in *'aktiviert'* über. Anders verhält es sich, wenn der Vaterprozeß niedrige Priorität hat, dem neuen Prozeß beim Anmeldevorgang mittels 'runp' jedoch hohe Priorität zuteilt. Der nun hochpriorisierte neue Prozeß geht sofort in den Zustand 'aktiv' über.

Aktive Prozesse mit niedriger Priorität können vom 'Scheduler' deaktiviert werden, falls sie zwei Zeitscheibendurchläufe ununterbrochen gerechnet haben. Sie wer-

den dann wieder an die Liste der bereiten Prozesse angehängt und warten auf ihre Reaktivierung. Außerdem können sie von Prozessen hoher Priorität unterbrochen werden. Dies kann eintreten, wenn ein Prozeß hoher Priorität zuvor im Zustand 'wartend' war und in den Zustand 'bereit' übergegangen ist, oder wenn ein Prozeß hoher Priorität von einem Prozeß mit niedriger Priorität angemeldet wurde und dann vom 'Scheduler' sofort aktiviert wird. Prozesse hoher Priorität können nicht unterbrochen werden. Für einen *unterbrochenen* Prozeß niedriger Priorität gibt es eine spezielle Datenstruktur, die die Registerinhalte sichert, so daß dieser sofort nach der Beendigung des priorisierten Prozesses wieder aktiviert werden kann, ohne den Umweg über die Bereitschaftsliste zu gehen. Weiterhin können aktive Prozesse gewollt beendet werden, indem der 'Scheduler' durch Aufruf spezieller Maschinenbefehle dazu veranlaßt wird, den Prozeß in den Zustand *'angehalten'* zu überführen. Dieser Zustand kann bei der Fehlersuche in Programmen nützlich sein. Der 'Scheduler' 'vergißt' sozusagen den angehaltenen Prozeß, da solche Prozesse nicht in einer Liste geführt werden. Normalerweise terminieren Prozesse nach der Ausführung des Maschinenbefehls 'endp'. Sie gehen in den Zustand *'beendet'* über. Der 'Scheduler' verwaltet beendete Prozesse nicht länger in der Prozeßliste.

Aktive Prozesse, die auf ein Ereignis warten, gehen in den Zustand *'wartend'* über. Es gibt grundsätzlich drei verschiedene Ereignisse, auf die ein Prozeß warten kann.

1. Es ergibt sich ein Rendezvous. Ein Prozeß, der auf einen OCCAM-Kanal schreiben oder von ihm lesen will, wird vom 'Scheduler' solange in den Zustand 'wartend' versetzt, bis der entsprechende Sende- bzw. Empfangsprozeß ebenfalls bereit ist zu kommunizieren.

2. Eine programmierte Zeitdauer ist abgelaufen. Ein Prozeß kann sich selbst für eine bestimmte Zeit in den Zustand 'wartend' überführen, indem er ein Zeitoffset[5] beim Zugriff auf einen Zeitgeber (s.u.) angibt. Nach Verstreichen der dem Zeitoffset entsprechenden Zeit benachrichtigt der Zeitgeber den 'Scheduler', daß der entsprechende Prozeß wieder in die Bereitschaftsliste aufzunehmen ist.

3. Ein Prozeß, der auf einen Interrupt reagieren soll, wird durch den Interrupt aktiviert. Ein Prozeß, der von dem speziellen Kanal *'Event'* lesen will, wird automatisch zum 'Interrupt-Bearbeitungs-Prozeß'. Mit anderen Worten: Das Lesen vom 'Eventkanal' bindet den lesenden Prozeß an den Interrupt des Transputers.

3.2.1.3 Zeitgeberlogik

Die Transputer verfügen über zwei unabhängige Zeitgeber mit einer Auflösung von 1μs und 64μs. Alle Prozesse können diese Zeitgeber abfragen. Implementiert sind sie als 32 Bit - Register. Alle 64μs bzw. 1μs werden diese Register um 1 erhöht. Ist die größte positive Zahl (in der Abbildung 3.5 als MAX_POS_INT bezeichnet) erreicht, wird

[5] Ein Zeitoffset ist eine positive Zahl, die die Anzahl der nötigen Erhöhungen des Zeitgeber- Registers (s.u.) angibt. In Abhängigkeit von der Auflösung des Zeitgebers beträgt ein Zeitoffset von 1 entweder 1μs oder 64μs.

bei der kleinsten negativen Zahl (MIN_NEG_INT) fortgefahren. Der Zahlenbereich wird also zyklisch durchlaufen. Ein Durchlauf dauert: $1\mu s * 2^{32} = 1,193046471h \approx 1h11min35sec$. Bei $64\mu s$ verlängert sich ein solcher Durchlauf entsprechend auf 3 Tage 4 h 21 min 18 sec.

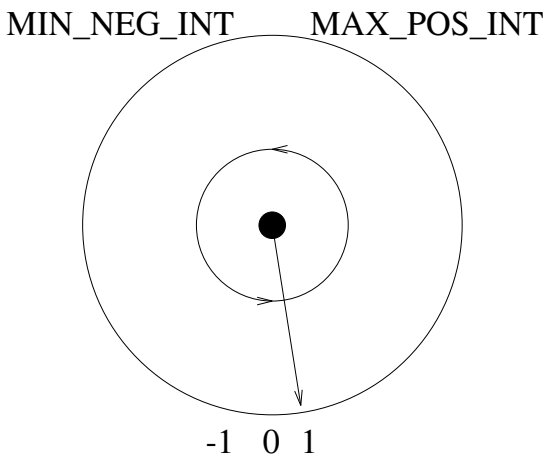


Abbildung 3.5: Wertebereich der Transputer-Zeitgeber (vgl. [Ebe93, Seite 454])

3.2.1.4 'Link-Schnittstellen'

Jeder Transputer verfügt über vier unabhängige 'Link-Schnittstellen' (kurz : Links), die eine Vernetzung mit anderen Transputern erlauben. Bei der Kommunikation über die Links wird die CPU nicht belastet, da jeder Link über eine eigene 'Direct Memory Access' (DMA) - Schnittstelle verfügt, also eigenständig auf Quelle oder Senke der Datenübertragung im Speicher zugreifen kann. Die Verbindung zweier Transputer über Links ist symmetrisch in dem Sinn, daß der Ausgang eines Links an den Eingang eines anderen geführt wird und umgekehrt. Die Brutto-Datenrate für die Linkgeschwindigkeit ist einstellbar. Sie kann zwischen 5Mbit/s, 10Mbit/s und 20Mbit/s gewählt werden. Über die Links wird sequentiell kommuniziert. Ein zu versendendes Byte wird mit zwei Startbits und einem Stopbit versehen. Jedes dieser 11bit langen Pakete wird vom Empfänger bestätigt.

Diese Bestätigung ist bei den Transputern (T800) im Arbeitsbereich Rechnerorganisation der Universität Hamburg (RO) überlappend. Ein Paket wird schon nach den ersten 6 Bit bestätigt, so daß die Sendeseite die Bestätigung empfängt, wenn das letzte Bit versendet wurde. Es treten also keine Sendeverzögerungen für aufeinanderfolgende Pakete auf. Damit ergibt sich die theoretische Obergrenze für die Netto-Datenrate, also die Nutzinformationen ohne Start– und Stopbits, zu $(\frac{20Mbit}{s}/11\frac{bit}{Byte})/1024 = 1775,568$Kbyte/s für die unidirektionale Kommunikation. Für die bidirektionale Kommunikation kann dieser Wert leider nicht verdoppelt werden, da sich hier die Pakete

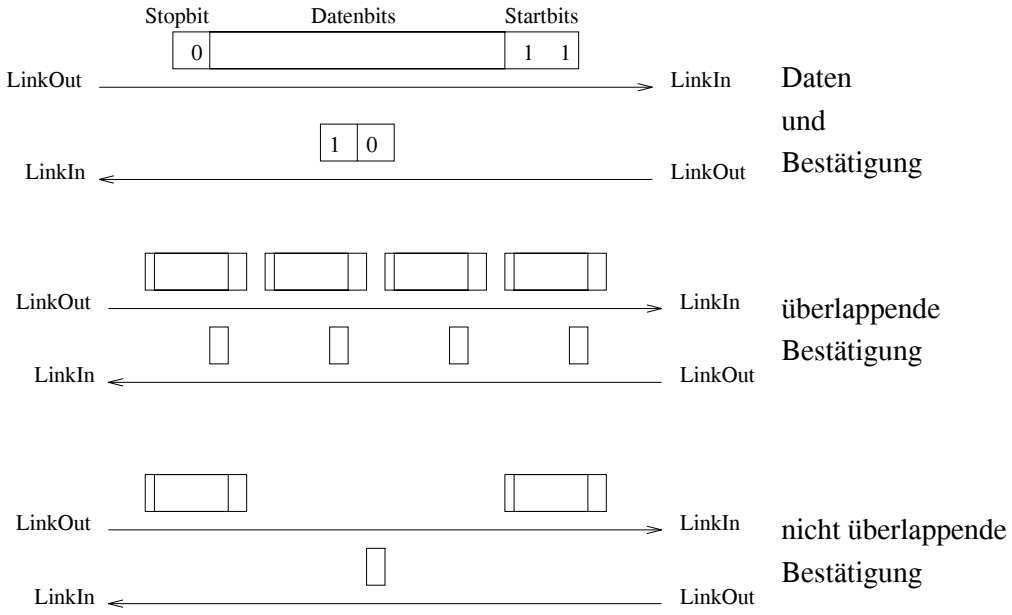


Abbildung 3.6: Protokoll der 'Link-Schnittstellen' der Transputer (vgl. [Ebe93, Seite 445])

und die jeweiligen Bestätigungen die Übertragungskapazität teilen müssen.

In eigenen Messungen konnte gezeigt werden, daß aufgrund dieses Protokolls bei unidirektionaler Kommunikation die maximale Übertragungsgeschwindigkeit 1472,8 Kbyte/s beträgt. Bei bidirektionaler Kommunikation beträgt die Übertragungsgeschwindigkeit 2137,25 Kbyte/s (vgl. [Ben94, S. 102ff]).

3.2.1.5 Transputer-Boards

Transputer werden zusammen mit lokalem Speicher, der an die externe Speicherschnittstelle angeschlossen wird (vgl. Abbildung 3.3), auf 'Transputer-Boards' (TBs) angeordnet. Außerdem befinden sich auf TBs spezielle Baugruppen für die Anbindung der Transputer an den 'Hostrechner' (s.u.) und an Peripheriegeräte bzw. für die Koppelung der Transputer untereinander. Im Arbeitsbereich RO werden zwei verschiedene TB-Typen eingesetzt.

MTM-PC-Board (Parsytec)

Wesentliche Merkmale dieses TB sind:

- vier T800 Transputer
- vier Megabyte lokaler Speicher pro Transputer

- Konfigurationskanäle für die Vernetzung mehrerer TB
- programmierbare Kreuzschienenverteiler
- Busschnittstelle für IBM-PC-kompatible Rechner

Durch den programmierbaren Kreuzschienenverteiler sind diese TBs besonders flexibel konfigurierbar. Die Links der Transputer lassen sich so beliebig untereinander verbinden. Außerdem kann man die Links durch eine entsprechende Programmierung des Kreuzschienenverteilers auch mit Buchsen auf dem TB verbinden, so daß eine TB-übergreifende Verbindung zwischen Transputern hergestellt werden kann. Damit hat man die Möglichkeit, beliebig viele Transputer miteinander zu vernetzen.

Der Hauptvorteil der programmierbaren Kreuzschienenverteiler ist jedoch, daß sich die Verbindungen der Transputer zur Programmlaufzeit dynamisch verändern lassen. Man kann so sehr flexibel auf neue Anforderungen reagieren. In Abbildung 3.7 ist der Aufbau eines 'MTM-PC-Boards' schematisch dargestellt. Der mit einem Stern markierte Transputer unterscheidet sich von den anderen dadurch, daß zwei seiner Links fest verbunden sind. Ein Link ist mit der PC-Schnittstelle verbunden. Hierüber muß die gesamte Transputer-PC-Kommunikation ablaufen. Ein weiterer Link ist mit dem Kreuzschienenverteiler fest verbunden. Über diesen Link wird der Kreuzschienenverteiler programmiert.

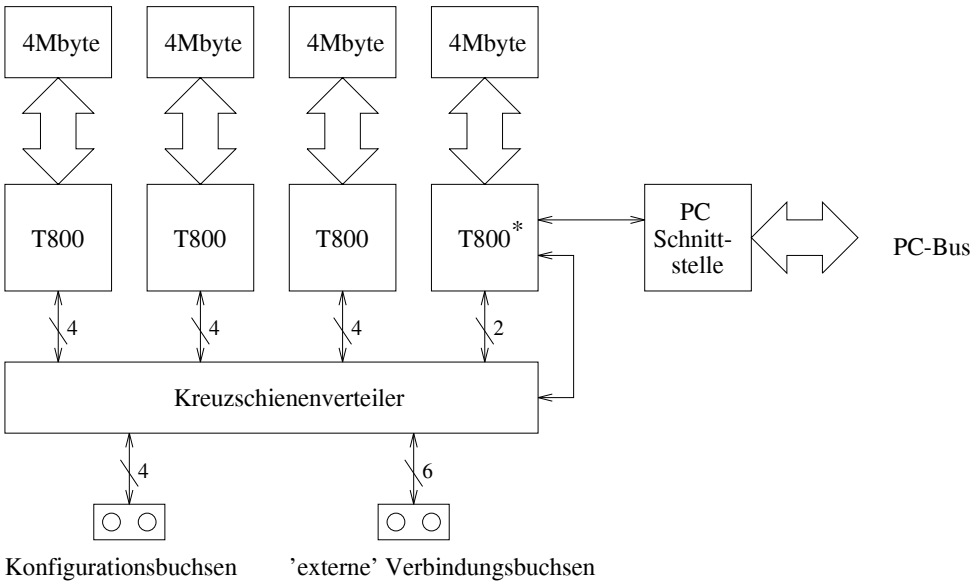


Abbildung 3.7: Struktur eines 'MTM-PC-Board'

INMOS-kompatibles-Board

Auch dieses TB verfügt über eine Busschnittstelle für IBM-PC-kompatible Rechner und über Buchsen für eine TB-übergreifende Vernetzung der Transputer.

Auf diesem TB ist jedoch nur ein einzelner T225 Transputer untergebracht, der über gemeinsamen Speicher an eine Ethernet-Schnittstelle gekoppelt ist. Damit ist es möglich, von den Transputern direkt[6] auf ein Kommunikationssystem zuzugreifen.

In Abbildung 3.8 ist dieses TB schematisch dargestellt.

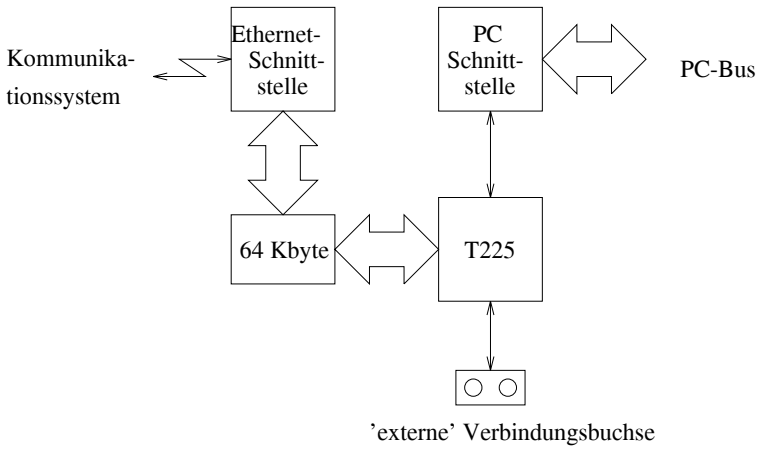


Abbildung 3.8: 'INMOS-kompatibles-Board' mit Ethernet-Schnittstelle

3.2.1.6 Koppelung mit dem 'Hostrechner'

Die oben beschriebenen TB werden in IBM-PC-kompatible Rechner (PCs) eingesetzt. Ein solcher PC dient als 'Hostrechner' und stellt den Transputern seine Peripherie zur Verfügung. Insbesondere sind zu nennen:

- Tastatur für Benutzereingaben,
- Bildschirm für Ausgaben,
- Massenspeicher (Disketten, Festplatten etc.) zum Speichern großer Datenmengen.

Die Ankoppelung TB an PCs erfolgt über die oben genannten Busschnittstellen. Diese setzen die über einen Link seriell übertragenen Datenpakete (vgl. Abbildung 3.6) in ein Byte lange Wörter um. Diese Byte-Sequenz kann der PC auslesen.

[6] Indirekt könnten die Transputer auch über eine Schnittstelle des 'Hostrechners' auf ein Kommunikationssystem zugreifen. Effizienter ist jedoch der direkte Zugriff.

Eine entsprechende Umsetzung von Ein-Byte-Wörtern in Link-konforme Datenpakete ermöglicht es dem PC, Daten an die Transputer zu versenden. Für diese bidirektionale PC-Transputer-Kommunikation werden von den TBs vier Ein- Ausgabeadressen des PC belegt. Zwei dieser Adressen dienen als Datentore für die Ein-Byte-Wörter. Es gibt jeweils eine Adresse für jede Übertragungsrichtung. Über die beiden anderen Adressen hat der PC Zugriff auf Statusinformationen, die anzeigen, ob der Transputer bereit ist, weitere Daten zu empfangen bzw. weitere Daten zu senden.

Durch diesen Mechanismus ist ein einfacher Abfragebetrieb[7] möglich. Zusätzlich kann man Daten vom Transputer "interrupt-gesteuert" empfangen. Dadurch entfällt das ständige Abfragen der Statusinformation, um Nachrichten vom Transputer zu empfangen.

3.2.1.7 Gesamtarchitektur der Hardware

Die 'Hostrechner' werden nicht gesondert dargestellt, da es sich um weit verbreitete IBM-kompatible PCs handelt. Diese PCs sind beliebig gegen gleichwertige oder bessere Hardware austauschbar. Die Mindestanforderung an einen 'Hostrechner' ist ein Intel 486 Prozessor, da die Entwicklungsumgebung (s.u.) diesen voraussetzt.

In Abbildung 3.9 ist die gesamte 'Hardware-Architektur' schematisch dargestellt. Auf Grundlage dieser 'Hardware-Architektur' werden in Abschnitt 4.1 die Restriktionen durch die Hardware beschrieben.

[7] Durch ständiges 'pollen' kann der PC prüfen, ob der Transputer Daten senden möchte oder bereit ist, selber Nachrichten zu empfangen.

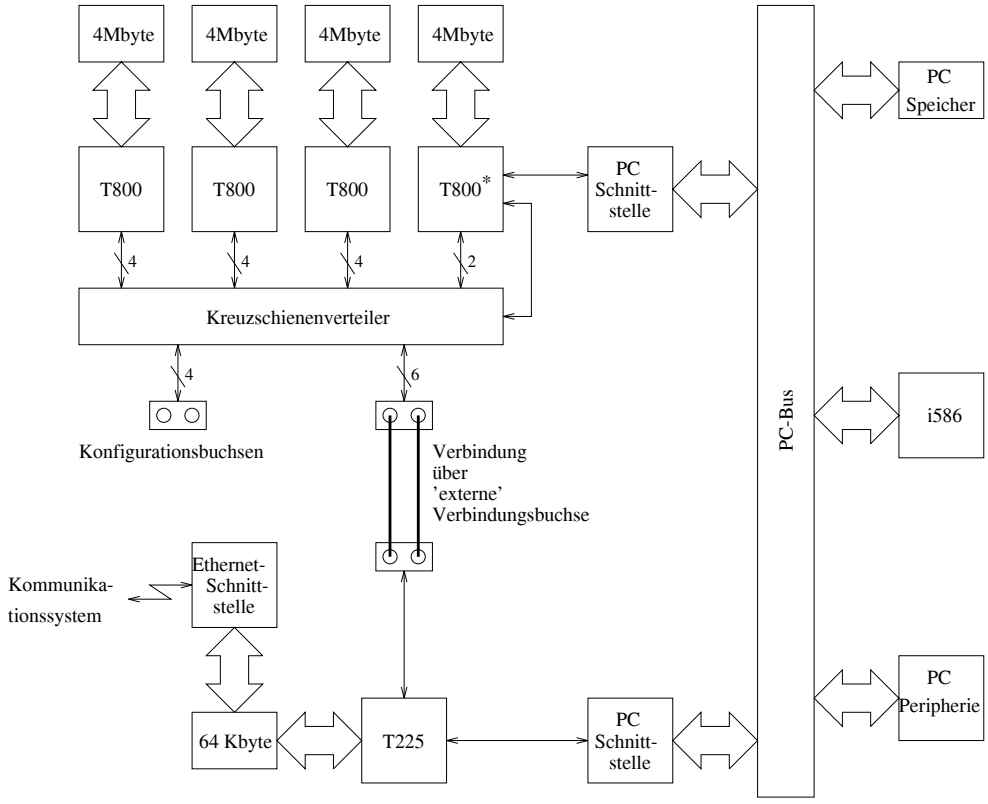


Abbildung 3.9: 'Hardware-Architektur'

3.2.2 Entwicklungsumgebung

Für die Programmerstellung werden zwei verschiedene Entwicklungsumgebungen eingesetzt. In beiden Fällen wird die Programmiersprache 'C' benutzt. Wegen der weiten Verbreitung dieser Sprache ist so eine gute Portierbarkeit auf andere 'Hostrechner' sichergestellt. Außerdem brauchen verschiedene Programmodule nur einmal erstellt werden und können dann sowohl auf dem PC als auch auf den Transputer eingesetzt werden[8]. Dies wäre bei zwei verschiedenen Programmiersprachen nicht möglich.

Prozesse, die auf dem 'Hostrechner' plaziert werden sollen, müssen mit Übersetzern[9] erstellt werden, die Programmcode für den Prozessor des 'Hostrechners' erstellen (vgl. Abschnitt 3.2.2.1). Die Entwicklungsumgebung für die Transputer wird in Abschnitt 3.2.2.2 beschrieben.

[8] Beispiele für solche universellen Module sind abstrakte Datentypen wie Warteschlangen, Stapel usw.

[9] engl. 'Compiler'

3.2.2.1 Entwicklungswerkzeuge für den 'Hostrechner'

Als Übersetzer für den PC wurde der weit verbreitete GNU-C Compiler gewählt. Mit Ausnahme der systemspezifischen Funktionen können die Quellprogramme durch einfaches Neuübersetzen auf einen anderen Rechnertyp übertragen werden, da der GNU-C Compiler für alle relevanten Systeme[10] (frei) verfügbar ist.

Auf dem PC wird OS/2 Version 3.0 als Betriebssystem eingesetzt. Gegenüber MS-Dos bietet OS/2 ein präemptives Scheduling für Prozesse und eine graphische Benutzeroberfläche. Damit vereinfacht sich sowohl die Programmierung (Verteilen der Funktionen auf Prozesse) als auch die Präsentation der Analyseergebnisse.

Auf dem PC werden darüber hinaus folgende Werkzeuge eingesetzt:

- der 'GNU-Debugger' zur Fehlersuche,
- 'Fast-Lex' ein 'Scanner-Generator',
- 'GNU-Bison' ein 'Parser-Generator',
- 'GNU-make' zur Projektverwaltung,
- 'GNU-RCS' zur Versionsverwaltung der Quelltexte.

Auch diese Werkzeuge sind für alle relevanten Systeme (frei) verfügbar.

3.2.2.2 Entwicklungsumgebung für die Transputer

Alle Programme für die Transputer werden mit einem speziellen C-Compiler erstellt[11]. Neben dem üblichen ANSI-C-Sprachumfang stellt er transputerspezifische Funktionen zur Verfügung. Durch diese Erweiterungen ist es möglich,

- Prozesse zu generieren und zu synchronisieren,
- die Kanalkommunikation der Transputer zu benutzen,
- die Zeitgeber der Transputer abzufragen.

Außerdem werden für die Programmentwicklung benötigt:

- 'iserver' zum Laden der Prozesse in ein Transputernetz,
- 'config' zum Beschreiben der Transputernetze und Erstellen von 'Ladeprogammen'[12] für 'iserver'.

Leider verfügen wir nicht über einen 'Debugger' für die Transputer. Die Fehlersuche in den nebenläufigen Algorithmen ist dementsprechend aufwendiger.

[10] Es existieren Anpassungen für praktisch alle Unix-Derivate, MS-Dos und OS/2 sowie für VMS-Rechner. Es werden außerdem diverse andere Betriebssysteme unterstützt, die aufgrund ihrer geringen Verbreitung hier nicht genannt sind.

[11] Es handelt sich hierbei um 'Par-C' der Firma 3L Ltd.

[12] 'iserver' startet nur den ersten Transputer auf einem TB über die PC-Schnittstelle (vgl. Abbildung 3.7) und übergibt diesem die Ladeprogramme für die anderen Transputer, die über das sogenannte 'Bootstrap' - Verfahren sukzessive gestartet werden.

3.3 Konzepte zur parallelen Protokollverarbeitung

Viele Meßaufgaben (vgl. Abschnitt 2) beziehen sich direkt oder indirekt auf die verwendeten Protokolle bei der Datenübertragung. Offensichtlich müssen die Dateneinheiten identifiziert und danach ihre genaue Zusammensetzung ('Protocol-Data-Unit-Format' kurz: PDU-Format) analysiert werden, um beispielsweise Statistiken zur Längenverteilung der PDUs zu erzeugen.

Es liegt daher nahe, ein geeignetes Konzept für die Analyse der Dateneinheiten aus dem Bereich der *parallelen Protokollverarbeitung* auszuwählen und es direkt, oder nach noch zu diskutierenden Modifikationen, für die Realisierung des Meßwerkzeuges zu verwenden. Bewertungskriterien, die eine Auswahl nachvollziehbar machen, werden in Abschnitt 3.3.1 vorgestellt.

Modifikationen, die durch die Adaption eines Konzeptes für die parallele Protokollverarbeitung an die parallele Protokollanalyse notwendig werden, ergeben sich aus den verschiedenen 'Sichtweisen' auf das Kommunikationssystem, die man bei der Protokollverarbeitung in Endsystemen im Gegensatz zu der Protokollanalyse mit einem Werkzeug hat. Eine Gegenüberstellung dieser Sichtweisen wird in Abschnitt 3.3.2 gegeben.

In Abschnitt 3.3.3 werden schließlich die verschiedenen Ansätze zur parallelen Protokollverarbeitung vorgestellt und in Abschnitt 3.3.4 für einen Einsatz in der parallelen Protokollanalyse bewertet.

3.3.1 Bewertungskriterien

Die Grundlage für die Bewertung der Konzepte in den folgenden Abschnitten sind:

- **Realisierbarkeit der funktionalen Anforderungen**. Die allgemeinen Anforderungen an ein Werkzeug wurden in Abschnitt 2 diskutiert, die speziell zu erfüllenden Anforderungen in Abschnitt 2.7. Das gewählte Konzept muß grundsätzlich die Implementierung der erforderlichen Funktionen zulassen. Neben dieser allgemeinen Eignung muß weiterhin beachtet werden, mit welchem *Aufwand* die Realisierung verbunden ist. Der Aufwand ist im Vergleich mit alternativen Konzepten abzuschätzen.

  Ein mögliches Maß für diesen relativen Aufwand wäre die Summe der zusätzlichen *Unterstützungsfunktionen*, da sich Realisierungen unterschiedlicher Konzepte gerade in diesen Unterstützungsfunktionen unterscheiden. Unterstützungsfunktionen erbringen Dienste für die Funktionen, die die eigentlichen (Benutzer-) Anforderungen realisieren. Dies sind beispielsweise Funktionen für die Inter- und Intraprozeßkommunikation und Funktionen für Warteschlangen mit bestimmten Bedienstrategien usw.

- **Erweiterbarkeit**. Der Aufwand bei Erweiterungen ist ein weiteres Kriterium für die Wahl eines Konzeptes. Beispielsweise könnte man den Aufwand bewerten, der entsteht, um die Analyse auf ein weiteres zu untersuchendes Protokoll auszudehnen.

Da ein konsequenter modularer Aufbau der Implementierung eine Erweiterung erleichtert, wäre eine andere Bewertungsmöglichkeit für die Erweiterbarkeit, ob das jeweilige Konzept eine funktionale Gruppierung zu Moduln zuläßt (vgl. Abschnitt 3.1).

Zusätzlich kann man sich fragen, ob bzw. wann das jeweilige Konzept die Erweiterung der Hardware erfordert. Können neue Funktionen als Prozesse auf die vorhandenen Prozessoren plaziert werden, oder müssen für neue Funktionen weitere Prozessoren angeschafft werden? Wie groß ist der Aufwand für die Rekonfiguration bei Hardware-Erweiterungen?

- **Leistungsfähigkeit**. A priori muß eine Realisierung die Benutzeranforderungen erfüllen können. Aus den funktionalen Anforderungen ergeben sich dann direkt Anforderungen an die Leistungsfähigkeit. Beispielsweise ist die maximale Anzahl zu analysierender Dateneinheiten pro Sekunde eine dieser Leistungsanforderungen, die sich direkt aus den zugrundeliegenden Netzeigenschaften ergeben. Bei Anpassungen und Erweiterungen an andere Netze (in Zukunft Hochgeschwindigkeitsnetze) steigen die Anforderungen an die Leistungsfähigkeit. Das jeweilige Konzept sollte auch zukünftigen Leistungsanforderungen genügen, mit anderen Worten, das Konzept muß *skalierbar* sein (vgl. Abschnitt 2.7.2 und 2.7.3).

- **Verträglichkeit mit der Hardware**. In Abschnitt 3.2 wurde die zu verwendende Hardware (HW) beschrieben. Ein Konzept sollte so gewählt werden, daß es nicht nur trotz bestimmter Restriktionen durch die HW realisierbar ist, sondern vielmehr die Vorzüge und Eigenschaften der jeweiligen HW nutzen kann. Beispielsweise wäre ein Konzept, das von der Speichersynchronisation der parallelen Prozesse ausgeht, ungeeignet, um auf der vorgestellten HW realisiert zu werden[13].

3.3.2 Vergleich zwischen Endsystem und Monitor

Bei dem Versuch, ein Konzept der parallelen Protokollverarbeitung für die parallele Analyse von (Protokoll-) Dateneinheiten einzusetzen, muß man die verschiedenen Sichtweisen auf das Kommunikationssystem kennen.

Ein Vergleich zwischen einem am Kommunikationssystem angeschlossenen Endsystem einerseits und einem ebenfalls am Kommunikationssystem angeschlossenen Meßwerkzeug (Monitor) andererseits, soll die unterschiedlichen Sichtweisen verdeutlichen.

[13] Wie in Abschnitt 3.2 dargestellt wurde, beruht die Interprozessorkommunikation der Transputer auf Nachrichtenaustausch über Kommunikationskanäle und nicht auf Synchronisationsmechanismen im gemeinsamen Speicher.

Endsystem	Monitor
Empfängt nur Dateneinheiten mit eigener Empfängeradresse.	Muß potentiell alle Dateneinheiten im untersuchten System empfangen können. Kann gegebenenfalls ein anderes System 'simulieren'.
'Normale' Protokollverarbeitung. Die Analyse der PDUs dient zur Rückgewinnung der eingekapselten Nachrichten.	Protokollverarbeitung zur statistischen, syntaktischen und semantischen Analyse von PDUs. Analyse des zeitlichen Ablaufs ('Protokoll-Timing').
Protokollhierarchie zur Strukturierung von Diensten.	Protokollhierarchie als (Teil-) Menge zu analysierender Protokolle.
Verarbeitet in der Regel die Protokolle einer Protokollfamilie.	Analysiert verschiedenartige Protokolle und Protokollfamilien.
Protokollautomaten werden als Prozesse realisiert.	Analyse braucht nicht durch Protokollautomaten realisiert werden.

Der Vergleich zeigt, daß sowohl das Endsystem als auch der Monitor die PDUs hierarchisch auswerten. Diese hierarchische Auswertung der Informationen erfolgt nach der verwendeten Protokollhierarchie.

Die Motivation für diese Hierarchisierung im Endsystem ist die Strukturierung und Bereitstellung der Dienste, wobei Dienste einer höheren Schicht die einer niedrigeren benutzen können, um die eigene Dienstleistung einer höheren Schicht zur Verfügung zu stellen.

Der Monitor muß ebenfalls diese hierarchische Auswertung durchführen, da nur durch sukzessives Erkennen eines Protokolls der nächsthöheren Schicht nach dem Empfang einer Dateneinheit entschieden werden kann, welche Bedeutung[14] die PDU-Daten haben.

Die Auswertung der PDUs gemäß einer Protokollhierarchie ist die Grundlage für die Adaption von Methoden der *parallelen Protokollverarbeitung* für die parallele Protokollanalyse.

3.3.3 Klassifikation der parallelen Protokollverarbeitung

Eine mögliche Klassifikation der parallelen Protokollverarbeitung geht von der hierarchischen Schichtenstruktur des 'Open System Interconnection Reference Model' aus. Dieses Referenzmodell (OSIRM) beschreibt eine für Protokolle typische hierarchische Anordnung, die Grundlage für die verschiedenen Ansätze zur parallelen Protokollverarbeitung ist (vgl. Abschnitt 3.3.2).

Es lassen sich Methoden zur parallelen Protokollverarbeitung aus der

[14] Die Bedeutung ergibt sich aus dem PDU-Format. Dieses Format ist in einer entsprechenden Protokolldefinition festgelegt.

- statischen Struktur und der
- dynamischen Struktur

im Referenzmodell ableiten. Die statische Struktur des Referenzmodells (vgl. [Ker92]) beschreibt die Hierarchisierung der Protokolle, die Dienste, die die Protokolle erbringen, sowie die Struktur und Adressierung der Instanzen, die das Potential der Protokollverarbeitung in einem System bilden. Die sich hieraus ableitbaren Methoden der Parallelverarbeitung unterscheiden sich in der *funktionalen Dekomposition* (vgl. Abschnitt 3.1) der Aufgaben bei der Protokollverarbeitung.

Die dynamische Struktur des Referenzmodells (vgl. [Ker92]) beschreibt das Verhalten der Kommunikationspartner bei der Datenübertragung, z.B. das Aufbauen von Verbindungen oder das verbindungslose Übertragen von Daten. Die hierunter subklassifizierten Formen der Parallelverarbeitung gehen von den versendeten Dateneinheiten (Paketen) aus. Sie unterscheiden sich in der *Dekomposition der Daten(einheiten)*, die in einem Endsystem eintreffen bzw. von dort versendet werden.

In Abbildung 3.10 werden die weiter unten vorgestellten Parallelisierungsformen vorab kurz dargestellt.

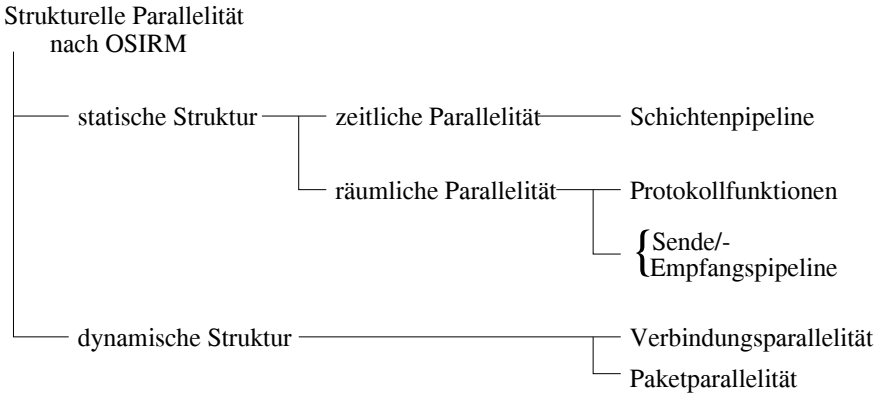


Abbildung 3.10: Strukturelle Parallelität nach OSIRM

3.3.3.1 Vertikale Parallelität (Protokollpipeline)

Die *vertikale Parallelität* ergibt sich aus der hierarchischen Struktur der Protokolle. Die Protokolle bilden eine *Protokollpipeline*. Auf den verschiedenen Schichten[15] arbeiten die Protokolle parallel zueinander. Wenn eine Protokollinstanz eine Dateneinheit verarbeitet hat, wird diese der folgenden Schicht übergeben. Dort beginnt die Protokollinstanz mit der weiteren Verarbeitung, während die Protokollinstanz der darunterliegenden Schicht eine weitere Dateneinheit bearbeiten kann (vgl. Abbildung 3.11).

[15] Bildlich gesprochen, sind die Schichten *vertikal* übereinander angeordnet.

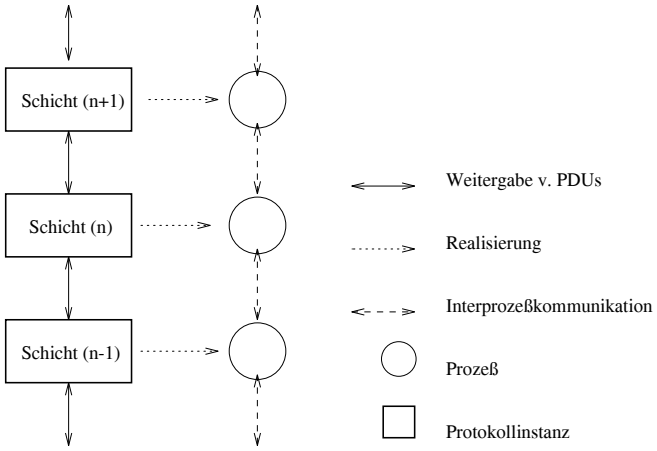


Abbildung 3.11: Protokollpipeline

3.3.3.2 Sende- und Empfangspipeline

Eine Sende- und Empfangspipeline (auch *horizontale Parallelität* genannt) entsteht durch die konzeptionelle Unterteilung jeder Protokollinstanz in ihre Sende- und Empfangsfunktionen. Diese (Gruppen von) Funktionen können weitgehend unabhängig voneinander — und somit parallel — ausgeführt werden (vgl. Abbildung 3.12).

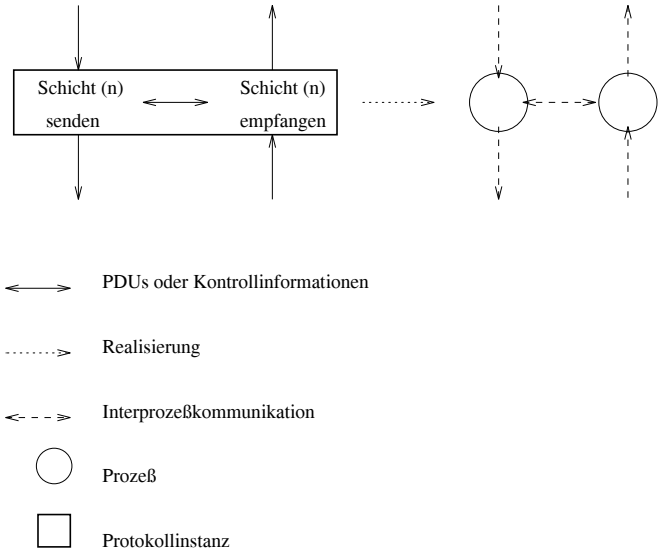


Abbildung 3.12: Sende/- Empfangspipeline

3.3.3.3 Protokollfunktionen

Will man eine feinere funktionale Dekomposition[16] erreichen als bei der Kombination aus horizontaler und vertikaler Parallelität[17], muß man die einzelnen Protokollinstanzen weiter unterteilen. Diesem Konzept liegt die Annahme zugrunde, daß in Protokollen verschiedene Funktionen unabhängig voneinander ausgeführt werden können. Diese Funktionen sind protokollspezifisch und müssen durch eine genaue Analyse des Protokolls gefunden werden (vgl. [Zit91b, S. 56f], Abbildung 3.13).

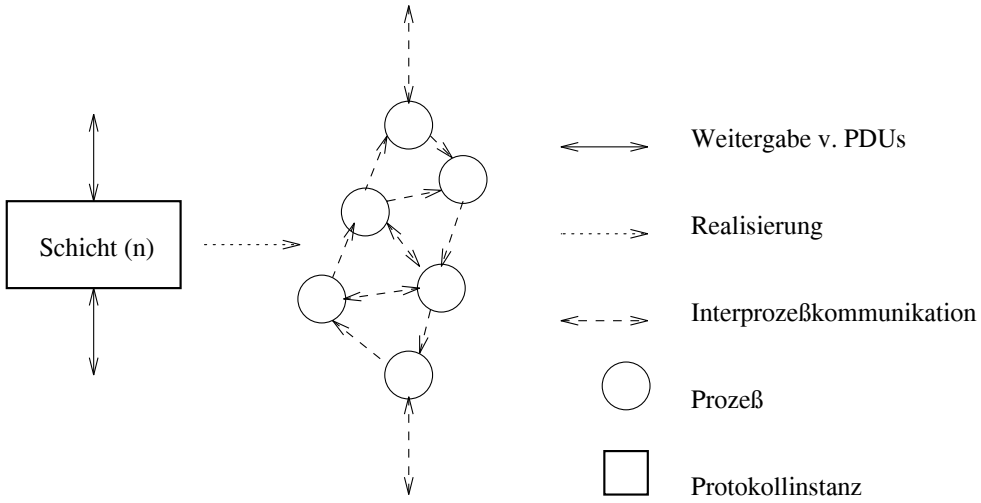


Abbildung 3.13: Protokollfunktionen (vgl. [Zit91b, S. 52ff])

3.3.3.4 Verbindungsparallelität

Die Verbindungsparallelität entsteht durch die verschiedenen Verbindungen, die in einem Endsystem aufgebaut werden können. Die Protokollverarbeitung kann i.d.R. für verschiedene Verbindungen völlig parallel erfolgen. Offensichtlich läßt sich der Grad der erreichbaren Parallelität nicht im voraus angeben. Vielmehr entspricht dieser exakt der Anzahl der zu einem Zeitpunkt aktiven Verbindungen in einem Endsystem. Die Granularität der Parallelität kann sich also dynamisch verändern (vgl. Abbildung 3.14 für eine schematische Darstellung).

[16] vgl. Abschnitt 3.1. Je feiner die funktionale Dekomposition ist, desto höher ist der erreichbare Grad an möglicher Parallelverarbeitung.

[17] Angenommen in einem Endsystem wird eine Protokollhierarchie aus n Protokollen eingesetzt, dann gibt es nach der funktionalen Dekomposition in horizontale und vertikale Parallelität genau $2n$ parallel ausführbare Funktionen, die als eigenständige Prozesse realisiert werden können.

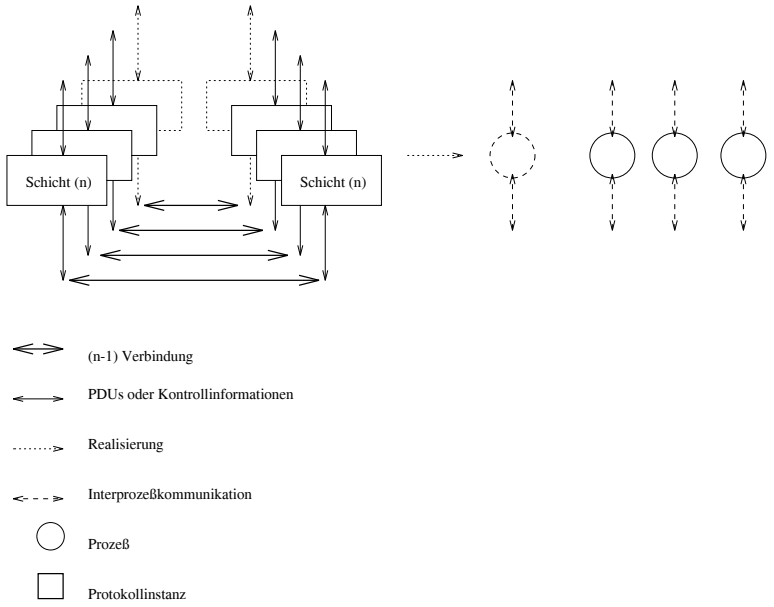


Abbildung 3.14: Verbindungsparallelität

3.3.3.5 Paketparallelität

Bei der Paketparallelität versucht man die Granularität gegenüber der Verbindungsparallelität zu erhöhen, indem man nicht mehr Verbindungen als kleinste unabhängige Einheiten ansieht, sondern die Dateneinheiten (Pakete) aller Verbindungen als voneinander unabhängig betrachtet. Jedes Paket kann konzeptionell unabhängig von allen anderen verarbeitet werden (vgl. [GNI93, S. 221ff] und Abbildung 3.15).

Für eine ausführliche Diskussion der Vor- und Nachteile der verschiedenen Ansätze vgl. [Zit91b], [WF93], [Ben94].

3.3.4 Bewertung der Konzepte zur parallelen Protokollverarbeitung

Die oben vorgestellten Konzepte zur parallelen Protokollverarbeitung können nun unter Berücksichtigung der Hardware (vgl. Abschnitt 3.2) für eine Anpassung an ein paralleles Meßwerkzeug bewertet werden.

Die Bewertungskriterien aus Abschnitt 3.3.1 sind:

- Realisierbarkeit der funktionalen Anforderungen,
- Erweiterbarkeit,
- Leistungsfähigkeit,
- Verträglichkeit mit der Hardware.

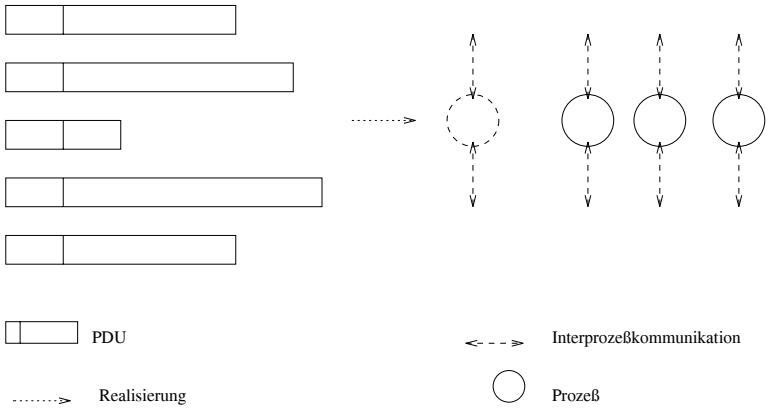


Abbildung 3.15: Paketparallelität

Die Bewertung anhand dieser Kriterien erfolgt durch grundsätzliche Aussagen zu den einzelnen Parallelisierungsmöglichkeiten, da diese in der Regel prinzipielle Vor- und Nachteile haben.

Darüber hinaus werden die Konzepte im Vergleich miteinander bewertet. Dies ist nötig, da sich die Vor- und Nachteile der einzelnen Konzepte schlecht gegeneinander 'verrechnen' lassen. Vielmehr muß der Vergleich der Konzepte zeigen, welche Punkte als besonders vor- oder nachteilig zu bewerten sind.

3.3.4.1 Bewertung der Schichtenparallelität

Das Konzept der Schichtenparallelität ist einfach zu realisieren. Für bereits implementierte Protokolle kann der sequentielle Programmtext übernommen werden. Die Anpassungen an ein Meßwerkzeug sind ebenfalls leicht realisierbar. Beispielsweise kann man protokollabhängige 'Trigger' der Pipelinestufe zuordnen, die der jeweiligen Protokollschicht entspricht.

Allerdings ist das Konzept der 'Protokollpipeline' für einige Protokolle unrealistisch. Beispielsweise sind die Protokolle IP und UDP[18] derart eng miteinander verzahnt, daß sie nur noch konzeptionell auf unterschiedlichen Protokollebenen stehen. Bei einer Realisierung würde man deshalb aus Effizienzgründen darauf verzichten, diese beiden Protokolle auf verschiedenen 'Pipelinestufen' zu verarbeiten.

Erweiterungen an der Protokollpipeline sind ebenfalls einfach realisierbar. Soll ein weiteres Protokoll analysiert werden, benötigt man eine weitere Pipelinestufe für dieses Protokoll. Veränderungen sind nur an der Stelle notwendig, wo das neue Protokoll identifiziert wird, also auf der jeweils davor liegenden Protokollschicht. Allerdings führen Erweiterungen um neue Protokollschichten unweigerlich zu Hardware-Erweiterungen, wenn man sich für eine isomorphe Zuordnung von Prozessen zu Prozessoren

[18] Die Abkürzungen stehen für das 'Internet Protocol' und das 'User Datagram Protocol'.

entschieden hat. In diesem Fall muß für jedes neue Protokoll, das als Prozeß realisiert wird, ein weiterer Prozessor zur Verfügung gestellt werden.

Positiv auf die Leistungsfähigkeit wirkt sich aus, daß die protokollabhängige Interprozeßkommunikation entfällt. Ein Prozeß bearbeitet vollständig ein Protokoll. So ist beispielsweise bei einem 'Protokolltester', der auf diesem Konzept beruht, keine Interprozeßkommunikation nötig, wenn die Sendekomponente auf Ereignisse der Empfangskomponente reagieren muß. Dies ist ein Vorteil gegenüber der Lösung gemäß Sende/-Empfangspipeline und dem Konzept der Protokollfunktionen (s.u.).

Negativ auf die Leistungsfähigkeit wirkt sich die geringe Granularität bei dem Konzept der Schichtenpipeline aus. Sollen beispielsweise n Protokolle analysiert werden, so ergeben sich maximal n Pipelinestufen und Prozesse, die auf maximal n Prozessoren plaziert werden können. Sind mehr als n Prozessoren vorhanden, können die übrigen Prozessoren nicht für die Protokollanalyse eingesetzt werden. Die geringe Granularität kann eventuell durch Kombination mit dem Konzept der Sende/-Empfangspipeline verbessert werden. Damit erhöht sich jedoch die protokollabhängige Interprozeßkommunikation (s.o.). Eine wesentlich feinere Granularität läßt sich mit dem Protokollfunktions-Konzept erreichen.

Zur Leistungsfähigkeit ist noch anzumerken, daß eine unausgewogene Pipeline (unterschiedliche Verarbeitungszeiten in den verschiedenen Pipelinestufen) zu einer suboptimalen Prozessorauslastung führt, da einige Prozessoren vor anderen fertig werden. Diese Unausgewogenheit ergibt sich bei einer Protokollpipeline zwangsläufig, da die Protokolle mit zunehmender Protokollschicht komplexer werden und somit der Analyse- und Rechenaufwand ebenfalls zunimmt. Protokollfunktionen, Verbindungsparallelität und Paketparallelität vermeiden diesen Nachteil.

3.3.4.2 Bewertung der Sende- und Empfangspipeline

Dieses Konzept ist ähnlich leicht zu realisieren wie das Konzept der Protokollfunktionen, da schon in sequentiellen Realisierungen meist zwischen Sende- und Empfangsfunktionen unterschieden wird. Eine Anpassung ist dementsprechend leicht. Dieses Konzept sollte jedoch nur in Kombination mit einer Schichtenpipeline verwendet werden, um eine möglichst feine Granularität zu erreichen.

Gegenüber einer reinen Schichtenpipeline ist dieses Konzept bei der Verarbeitung von verbindungslosen Protokollen leistungsfähiger. Da bei verbindungslosen Protokollen die protokollabhängige Interprozeßkommunikation zwischen Sende- und Empfangsprozeß minimal ist, kann die Parallelverarbeitung von den zwei verschiedenen Prozessen für die Sende- und Empfangsfunktionen profitieren.

Die Anpassung dieses Konzeptes für ein paralleles Meßwerkzeug lohnt außerdem nur dann, wenn neben der Protokollanalyse auch Protokolldateneinheiten vom Meßwerkzeug generiert werden sollen, da nur in diesem Fall Sendekomponenten in den Protokollschichten nötig sind. Dies ist beispielsweise bei einem 'Protokolltester' notwendig.

3.3.4.3 Bewertung der Protokollfunktionen

Im direkten Vergleich mit einer Sende/- Empfangspipeline kann man mit dem Konzept der Protokollfunktionen eine deutlich feinere Granularität erreichen. Aus der Sicht der Leistungsfähigkeit ist dieser Ansatz vorzuziehen. Die vorhandenen Prozessoren können durch eine ausgewogene Zuordnung der Protokollprozesse gleichmäßig belastet werden. Dadurch ist dieser Ansatz gut geeignet, um die Protokollverarbeitung einzelner Protokolle zu beschleunigen.

Allerdings sind bei diesem Ansatz eine Vielzahl von Unterstützungsfunktionen notwendig. Der anfallende 'Overhead' ist deutlich höher als bei Schichtenpipeline und Sende/- Empfangspipeline, da die protokollspezifische Interprozeßkommunikation aufgrund der Vielzahl von Prozessen pro Protokoll erheblich ist. Bei einer Zuordnung der vielen Prozesse auf Prozessoren ergibt sich weiterer 'Overhead' durch die vielen Kontextwechsel, die für den quasiparallelen Ablauf der Prozesse auf einem Prozessor nötig sind.

Aus der Sicht der Realisierbarkeit und der Erweiterbarkeit ist dieses Konzept völlig ungeeignet für eine Anpassung an die parallele Protokollanalyse. Das Auffinden der inhärenten Parallelität ist schon für ein einziges Protokoll aufwendig (vgl. [Zit91a]). Soll das Werkzeug ganze Protokollfamilien analysieren können, müßte bei diesem Konzept für jedes einzelne Protokoll eine eigene Analyse angefertigt werden. Ein derartiger Arbeitsaufwand ist nicht zu vertreten.

Die Verträglichkeit mit der Hardware muß u.U. erst hergestellt werden. In [Zit91a, S. 139] wird beschrieben, daß spezielle 'Kommunikationsprozesse' realisiert wurden, um die vielen Kommunikationskanäle der Protokollfunktion zu bündeln, da die Transputer nur vier Links haben. Diese Restriktion der Hardware konnte nur durch zusätzlichen 'Overhead' (nämlich durch die Kommunikationsprozesse) ausgeglichen werden.

3.3.4.4 Bewertung der Verbindungsparallelität

Die Verbindungsparallelität ist sehr leicht realisierbar und erweiterbar. Auf jedem Prozessor wird der gleiche Prozeß plaziert, so daß jeder Prozessor eine Verbindung unterstützen kann. Auch hier kann wie bei der Protokollpipeline der eventuell aus vorangegangenen sequentiellen Realisierungen vorhandene Programmcode bei einer Anpassung verwendet werden.

Der Hauptnachteil dieses Konzeptes ergibt sich aus der geringen Leistungsfähigkeit bei wenigen Verbindungen. Sind weniger Verbindungen als Prozessoren vorhanden, können die 'brachliegenden' Prozessoren nicht zur Beschleunigung beitragen.

Außerdem werden die Prozessoren im allgemeinen nicht gleichmäßig ausgelastet, da verschiedene Verbindungen typischerweise auch verschiedene Lasten generieren. Bei sich schnell auf- und abbauenden Verbindungen kann es außerdem dazu kommen, daß einzelne Prozesse mehrere Verbindungen verwalten, während andere, deren Verbindungen zwischenzeitlich abgebaut wurden, nicht ausgelastet sind. Diese drei Nachteile vermeidet der Ansatz der Paketparallelität (s.u.).

Allerdings gibt es bei der Verbindungsparallelität keine protokollspezifische Inter-

prozeßkommunikation, da verschiedene Verbindungen i.d.R. völlig unabhängig voneinander sind. Dennoch fällt 'Overhead' an, da die verschiedenen Prozesse nicht nur die Protokollanalyse leisten, sondern zuvor die Dateneinheiten 'ihrer Verbindung' selektieren müssen. Will man dies effizient realisieren, müßte man eintreffende Dateneinheiten in gemeinsamen Speicher legen, so daß alle Prozesse die Möglichkeit haben, nebenläufig auf Zugehörigkeit zu ihrer Verbindung zu prüfen. Die zu verwendene Hardware verfügt jedoch nicht über gemeinsamen Speicher.

Außerdem ist dieses Konzept ungeeignet, um Dateneinheiten von verbindungslosen Protokollen zu analysieren. Nur durch 'Verteilungs-Prozesse' und dem damit verbundenen 'Overhead' kann entschieden werden, welcher Prozeß sozusagen 'neben seiner Verbindung' eine solche Dateneinheit verarbeitet.

3.3.4.5 Bewertung der Paketparallelität

Das Konzept der Paketparallelität ist leicht realisierbar. Ähnlich der Verbindungsparallelität werden gleichartige Prozesse auf die Prozessoren plaziert. Auch hier kann eventuell vorhandener sequentieller Programmcode weiter benutzt werden. Alle für die Auswertung der Dateneinheiten notwendigen Funktionen werden zu einem Prozeß zusammengefaßt.

Erweiterungen sind ebenfalls leicht einzubringen. Der Prozeß wird um das neue Modul erweitert. Der Hauptvorteil dieses Konzeptes bei Erweiterungen ergibt sich aus der Gleichartigkeit der Analyseprozesse. Eine Erweiterung ist somit automatisch auf allen Prozessen präsent.

Ein Nachteil dieses Ansatzes besteht in den dezentral anfallenden Analyseergebnissen. Da Dateneinheiten des gleichen Protokolls von verschiedenen Prozessen analysiert werden können, muß man durch zusätzliche Unterstützungsfunktionen (mit entsprechendem 'Overhead') dafür sorgen, daß diese Ergebnisse zusammengeführt werden. Protokollabhängige Interprozeßkommunikation findet nur bei verbindungsorientierten Protokollen statt.

Die Leistungsfähigkeit ist unter hoher Last besonders gut. Alle Prozesse können durch einfaches Verteilen der eintreffenden Dateneinheiten annähernd gleichmäßig ausgelastet werden. Damit bietet dieser Ansatz bei deutlich geringerem Realisierungsaufwand eine zu den Protokollfunktionen vergleichbare Leistungsfähigkeit. Der Ansatz der Protokollfunktionen wäre nur bei niedriger Last (sporadisch eintreffende Dateneinheiten) überlegen.

Ein weiterer Vorteil dieses Ansatzes ist, daß für die Analyse auf unterschiedlichen Protokollschichten keine Kontextwechsel nötig sind, da derselbe Prozeß alle Protokolle analysiert.

Die Analyseprozesse können fast beliebig auf ein Prozessornetz plaziert werden. Aus Effizienzgründen wird man auch bei diesem Ansatz versuchen, die Analyseprozesse zusammenhängend auf Prozesse zu plazieren. Die Prozessortopologie ist jedoch nicht wie bei Protokollfunktionen oder Schichtenpipeline festgelegt[19]. Vielmehr läßt sich

[19] Es wird die Annahme unterstellt, daß man bei Protokollfunktionen und Protokollpipeline versucht, eine isomorphe Zuordnung zu finden.

dieser Ansatz auch auf völlig reguläre Prozessornetze abbilden.

Ein weiterer Vorteil im Zusammenhang mit der Hardware ist, daß Erweiterungen der Analysefunktionalität nicht notwendigerweise auch zu Hardware- Erweiterungen führen. Außerdem ist dieser Ansatz unabhängig von der Anzahl der verwendeten Transputer. Schon für wenige Prozessoren lohnt sich das Konzept der Paketparallelität. Dies ist ein Vorteil gegenüber der Schichtenpipeline, bei der mindestens so viele Prozessoren vorhanden sein sollten, wie zu analysierende Protokollschichten.

Kapitel 4

Entwurf des Meßwerkzeuges

In diesem Kapitel wird das Meßwerkzeug entworfen. Dazu werden in Abschnitt 4.1 die Restriktionen durch die Hardware beschrieben, da durch diese Restriktionen die Freiheiten beim Entwurf festgelegt werden. Anschließend wird eines der Konzepte der parallelen Protokollverarbeitung für eine Adaption an die parallele Protokollanalyse ausgewählt (vgl. Abschnitt 4.2).

Durch die Festlegung des Konzeptes zur parallelen Protokollanalyse kann dann in Abschnitt 4.3 die Grobgliederung des Meßwerkzeuges, unter Berücksichtigung der Restriktionen durch die Hardware, angegeben werden. Eine Verfeinerung der Grobgliederung führt in Abschnitt 4.4 zum Entwurf der *essentiellen* Funktionen. Diese Funktionen dienen direkt der Realisierung der Benutzeranforderungen und sind daher unverzichtbar für das Meßwerkzeug.

Neben den essentiellen Funktionen werden *unterstützende* Funktionen in Abschnitt 4.5 entworfen. Diese Funktionen unterstützen die essentiellen Funktionen bei ihrer Arbeit. Abschließend werden die Funktionen in Abschnitt 4.7 klassifiziert, um die Ergebnisse des Entwurfs zusammenzufassen, bevor in Kapitel 5 die Realisierung des Meßwerkzeuges dargestellt wird.

4.1 Restriktionen durch die Hardware

In Abschnitt 2.7.4 wurde bereits angedeutet, daß veraltete Hardware eingesetzt werden muß. Diese Hardware wurde in Abschnitt 3.2 vorgestellt, so daß hier die Restriktionen durch die Hardware besprochen werden können, die bei dem Entwurf und der Realisierung des Meßwerkzeuges beachtet werden müssen.

In der Darstellung der Hardware in Abschnitt 3.2 wurde deutlich, daß es zwischen 'Hostrechner' und den 'Transputer-Boards' (TBs) jeweils nur eine bidirektionale Verbindung gibt. Die Verbindung des zweiten TBs (vgl. Abbildung 3.8) kann außerdem nur zum Laden von Programmen, nicht aber zur PC-Transputer-Kommunikation, benutzt werden.

Eine weitere Restriktion ist, daß nur ein einziger Transputer mit dem T225 verbunden werden kann. Der Zugriff der anderen Transputer auf die Ethernet-Schnittstelle

kann nur indirekt erfolgen.

Weitere Restriktionen ergeben sich durch die Anzahl der Links pro Transputer. Hier stehen maximal vier Links pro Transputer zur Verfügung. Der mit einem Stern (*) markierte Transputer in Abbildung 3.7 verfügt nur über zwei freie Links. Damit sind schon bei nur vier Transputern nicht mehr alle direkt miteinander koppelbar.

Außerdem erkennt man auf der Abbildung, daß es pro TB genau vier Transputer gibt. Die zu wählende Konfiguration[1] sollte also immer ein Vielfaches von vier Transputern verwenden, um alle eingesetzten TBs voll auszulasten.

Die Restriktionen durch die Hardware lassen sich wie folgt zusammenfassen:

- Die gesamte Transputer-PC-Kommunikation muß über einen dedizierten Transputer erfolgen.
- Der Zugriff auf das Kommunikationssystem über die Ethernet-Schnittstelle muß über einen dedizierten Transputer erfolgen.
- Es kann jeweils nur ein Prozeß auf einen bidirektionalen Link der Transputer zugreifen. Alle Prozesse eines Transputers müssen mit maximal vier Links auskommen.
- Jedes TB (Typ wie in Abbildung 3.7) stellt genau vier Transputer zur Verfügung.

4.2 Wahl eines Konzeptes zur parallelen Protokollanalyse

In Abschnitt 3.3.3 wurden die verschiedenen bekannten Konzepte zur parallelen Protokollverarbeitung vorgestellt und bewertet.

In dieser Arbeit soll das Konzept der Paketparallelität für die parallele Protokollanalyse verwendet werden. Ausschlaggebend für diese Wahl ist die Leistungsfähigkeit des Konzeptes bei hoher Last und die vergleichsweise leichte Realisierbarkeit. Außerdem lassen sich Erweiterungen bei diesem Konzept besonders leicht durchführen. Durch diese Merkmale genügt das Konzept auch den Anforderungen an Leistungsfähigkeit und Skalierbarkeit. (vgl. Abschnitt 2.7.2 und 2.7.3).

Ein weiterer wichtiger Grund ist, daß durch kleine Änderungen die Paketparallelität z.B. in eine Protokollpipeline umgewandelt werden kann. Das könnte für einen Leistungsvergleich der beiden Ansätze interessant sein. Denkbar sind auch hybride Lösungen, bei der die Analyse verbindungsloser Protokolle durch die Paketparallelität erfolgt. Verbindungsorientierte Protokolle könnten durch dedizierte Prozesse analysiert werden, so daß eine Protokollpipeline entsteht.

[1] Eine Konfiguration besteht aus den Prozessoren und den Verbindungen der Prozessoren untereinander sowie einer Abbildung der Prozesse auf die Prozessoren.

4.3 Grobgliederung des Meßwerkzeuges durch Prozesse

Der Entwurf der Grobgliederung des Meßwerkzeuges ergibt sich aus den Anforderungen und dem gewählten Paradigma zur Parallelverarbeitung. In Abschnitt 3.1.1 wurde dargestellt, daß beim Zuordnen von Algorithmen zu Prozessoren die inhärente Parallelität aufgespürt werden muß, um dann die parallelen Prozesse auf Prozessoren abzubilden. Im folgenden Abschnitt wird begründet, wie viele Prozessoren für den Prototypen verwendet werden sollen. In Abschnitt 4.3.2 können dann die wesentlichen Prozesse des Meßwerkzeuges identifiziert werden. Diese Prozesse werden in Abschnitt 4.3.3 Prozessoren zugeordnet, so daß sich die Grobgliederung des Meßwerkzeuges ergibt.

4.3.1 Die Anzahl der Prozessoren

Für den Prototypen sollen sechs Prozessoren verwendet werden. Dies ist die minimale Anzahl an Prozessoren, die sich aus der Kombination eines 'Transputerboards' (TBs) aus Abbildung 3.7 und eines 'Boards' aus Abbildung 3.8 zusammen mit dem Prozessor des 'Hostrechners' ergibt.

Ausschlaggebend für diese Festlegung ist, daß der Prototyp nicht zu komplex werden darf, um die Realisierung innerhalb der für eine Diplomarbeit vorgesehenen Zeit zu ermöglichen. Durch ein weiteres 'Transputerboard' würden jedoch weitere vier Transputer zur Verfügung stehen, die das Konfigurieren des Prototypen deutlich verkomplizieren würden. Das ergibt sich aus der Tatsache, daß die Zuordnung manuell erfolgen muß, wie in Abschnitt 3.1.4 dargestellt.

4.3.2 Die wesentlichen Prozesse des Meßwerkzeuges

Die wesentlichen Prozesse ergeben sich aus

- den Benutzeranforderungen und
- dem Paradigma der 'Paketparallelität'.

Die Prozesse und ihre Aufgaben werden in den folgenden Abschnitten dargestellt.

4.3.2.1 Prozesse zur Datenerfassung

Grundsätzlich muß ein 'Monitor' Rohdaten erfassen können (vgl. Abschnitte 1.3.2 und 2.7.4). Die zu erfassenden Rohdaten sind in diesem Fall Ethernet-'Frames'. Die Ethernet-'Frames' müssen überprüft werden, um netz-spezifische Fehler erkennen zu können[2]. Um dem Benutzer einen Überblick über derartige Fehler geben zu können, sollte der Prozeß zur Rohdatenerfassung rudimentäre Statistiken über die erkennbaren Fehler führen.

[2]Bei Ethernet können beispielsweise Übertragungsfehler durch eine Prüfsumme erkannt werden.

Ein weiterer Grund, die Rohdatenerfassung als eigenständigen Prozeß zu realisieren, ist die Möglichkeit, durch replizieren dieser Prozesse[3] an verschiedenen Stellen im Netz, Rohdaten (verteilt) erfassen zu können. Man hätte so mehrere 'Rohdatenerfassungsmodule', wie beispielsweise in [K+95, S. 56] dargestellt. Auch eine Anpassung an andere Netze fällt bei einer Kapselung der hardwarenahen Funktionen in einen eigenen Prozeß leichter. Die Ansteuerung der netz-spezifischen Schnittstellen kann (prozess-)lokal erfolgen, um anderen Funktionen eine einheitliche Schnittstelle an die Datenerfassung anzubieten. Man erhält so einen Netz-'Server'-Prozeß.

4.3.2.2 Prozesse zur Analyse

Durch die Wahl der 'Paketparallelität' als Paradigma für die parallele Protokollanalyse ergibt sich, daß mindestens zwei Prozesse die erfaßten Rohdaten parallel auswerten sollten[4]. Die Funktionalität der 'Paketträger'[5] muß identisch sein, damit jedes erfaßte Rohdatum von einem beliebigen Paketträger ausgewertet werden kann.

Nur wenn die Paketträger über die gleiche Funktionalität verfügen, kann man die Entscheidung darüber, von welchem Paketträger eine Dateneinheit ausgewertet werden soll, unabhängig vom Inhalt der Dateneinheit treffen. Wollte man teilweise spezialisierte Paketträger realisieren, müßte ein Teil der Analyse außerhalb der Paketträger erfolgen, um entscheiden zu können, welche(r) Paketträger für die Analyse geeignet sind (ist).

Bei dem Prototypen sollen alle Paketträger über die gleiche Funktionalität verfügen. Man erreicht eine vollständige Kapselung aller Analysefunktionen, kann jedes Rohdatum von einem beliebigen Paketträger analysieren lassen und braucht bei Erweiterungen um neue Analysefunktionen keine 'Fallunterscheidungen' bezüglich des Typs eines Paketträgers treffen. Die Realisierung des Prototypen und zukünftige Erweiterungen werden so erleichtert.

4.3.2.3 Prozesse für globale Statistiken

Die Analyseergebnisse fallen dezentral bei den Paketträgern an. Durch diese Eigenschaft der Paketparallelität ergibt sich die Notwendigkeit, die dezentral anfallenden Daten vor der Weitergabe an den Benutzer zu sammeln, um globale Berechnungen durchführen zu können. Für das Bewerten der Analyseergebnisse (vgl. Abschnitt 1.3.2.3) kann es ebenfalls erforderlich sein, alle Analyseergebnisse zentral zusammenzuführen. Alle Berechnungen, die auf bisher angefallenen Analyseergebnissen durgeführt werden müssen, werden in einem Prozeß für die 'globale Statistik' gekapselt.

[3] Voraussetzung hierfür ist eine entsprechende Anzahl 'Meßfühler' (hier: Ethernet-Schnittstellen, wie in Abbildung 3.8 dargestellt).

[4] Bei nur einem 'Analyseprozeß' könnte man nicht von 'Paketparallelität' sprechen, da alle Dateneinheiten sequentiell von diesem einen Prozeß ausgewertet werden müßten.

[5] Die Analyseprozesse werden im folgenden als *Paketträger* bezeichnet.

4.3.2.4 Prozesse des 'Hostrechners'

Bei der Beschreibung der Hardware in Abschnitt 3.2 wurde dargestellt, daß die TBs in einen 'Hostrechner' (Host) eingesetzt werden müssen, da die TBs nicht über eigene Peripherie verfügen. Nur unter Benutzung der Peripheriegeräte des Hosts können die Anforderungen bezüglich der Interaktion mit dem Benutzer und dem Verwalten der Daten erfüllt werden (vgl. Abschnitt 2.7.4).

Es müssen daher Prozesse realisiert werden, die die Anforderungen an die Interaktion mit dem Benutzer realisieren. Dabei müssen die Standardschnittstellen des Betriebssystems des Hosts benutzt werden. Für die Interaktion mit dem Benutzer müssen folgende Funktionen realisiert werden:

- Eine textbasierte Ein- und Ausgabe. Die wichtigsten Kommandos sollte der Benutzer sowohl über eine Kommandozeile als Zeichenketten eingeben können als auch über die Selektion mit einer 'Maus'. Fehlermeldungen und andere Nachrichten des Werkzeuges sollten als 'Klar-Text-Meldungen' ausgegeben werden können.

- Eine graphische Ein- und Ausgabe. Die Vorteile einer graphischen Ausgabe liegt in der anschaulichen Präsentation umfangreicher Daten. Die prinzipiellen Möglichkeiten der graphischen Ausgabe wurden bereits in Abschnitt 2.4 verdeutlicht.

  Auch die graphische Eingabe kann die Interaktion mit dem Benutzer vereinfachen. Umfangreiche Selektionen aus vorgegebenen Möglichkeiten[6] über 'Dialoge' der graphischen Benutzeroberfläche sind eine Möglichkeit. Die Auswahl der wichtigsten Kommandos über ein 'Menü' mit einer Maus ist eine weitere graphische Eingabemöglichkeit.

Außerdem muß es Prozesse geben, die Analysedaten bei Bedarf auf Massenspeicher auslagern. Auch hier ist für den sicheren Betrieb eine enge Koppelung an die Funktionen des Betriebssystems notwendig. Vor der Ausgabe ist sicherlich eine Umwandlung in ein Standardformat erforderlich, um die Daten anderen Werkzeugen zugänglich zu machen (vgl. Abschnitt 2.5).

Eine weitere Aufgabe, die von einem Prozeß auf dem Host übernommen werden sollte, ist das Konfigurieren der Transputer und Einladen der Transputer-Prozesse in die Prozessoren. Das sollte bei jedem Start des Prototypen automatisch geschehen. Eine manuelle Konfigurierung durch den Benutzer ist weder komfortabel noch sinnvoll, da hierfür neben der Kenntnis der Konfiguration auch der Umgang mit entsprechenden Konfigurierungswerkzeugen vom Benutzer erwartet wird. Die Kenntnis der Konfiguration ist jedoch für die Verwendung des Prototypen nicht erforderlich. Daher bietet sich das automatische Konfigurieren und die Kapselung der dafür notwendigen Funktionen in einem Prozeß an.

[6] Derartige 'Multiple Choice'-Formulare kann man sich beispielsweise zur Auswahl der zu erfassenden Meßwerte durch den Benutzer vorstellen, wie es bereits in den Anforderungen spezifiziert wurde (vgl. Abschnitt 2.7.4).

4.3.3 Entwurf der Grobstruktur des Meßwerkzeuges

Die folgende Konfiguration berücksichtigt die oben dargestellten Restriktionen. Da vier Transputer auf einem TB vorhanden sind, werden diese zusammen mit dem PC-Prozessor und dem Transputer auf dem TB mit der Ethernet-Schnittstelle benutzt. Damit stehen insgesamt sechs Prozessoren für das Meßwerkzeug zur Verfügung.

Aus den Restriktionen geht hervor, daß es zum PC und zum TB mit der Ethernet-Schnittstelle jeweils genau eine Verbindung gibt. Daraus ergeben sich die Verbindungen der Prozessoren P1 zu P2 und P5 zu P6 (vgl. Abbildung 4.1). Die verbleibenden Prozessoren P3 und P4 können nun jeweils untereinander und mit P2 und P5 verbunden werden. Daraus ergibt sich die Topologie aus Abbildung 4.1.

Die Zuordnung der Prozesse ist weitgehend restriktiv. Es gibt mehrere qualitative Restriktionen. Offensichtlich können die Prozesse des 'Hostrechners' ausschließlich P1 zugeordnet werden, da P1 der einzige 'Hostrechner'-Prozessor ist. Die Prozesse zur Rohdatenerfassung lassen sich ebenfalls leicht zuordnen. Sie sollten auf Prozessoren in unmittelbarer Nähe zur Ethernet-Schnittstelle plaziert werden. Die unmittelbare Ansteuerung der Ethernet-Schnittstelle muß ohnehin von P6 aus erfolgen, da dieser Prozessor an die Ethernet-Schnittstelle gekoppelt ist (vgl. Abbildung 3.8). Das Zwischenspeichern der Rohdaten könnte auf P5 erfolgen, da dieser Prozessor über mehr Speicher verfügt als P6 (4MByte vs. 64KByte). Auf P5 ist demnach ein deutlich größerer Zwischenspeicher möglich.

Der Prozeß für die globale Statistik wird auf P2 plaziert. Das ist sinnvoll, da über diesen Prozessor ohnehin alle Ergebnisse zum PC weitervermittelt werden. Eine Plazierung dieses Prozesses auf einen anderen Prozessor als P2 würde mehr 'Overhead' für die Berechnung und Vermittlung der globalen Statistiken bedeuten.

Die Zuordnung der Paketträger-Prozesse ist relativ frei. Sie müssen (logisch) zwischen der Rohdatenerfassung einerseits und dem Prozeß für die globale Statistik andererseits liegen. Man könnte also jeweils einen Paketträger-Prozeß auf P2–P5 plazieren. Gegen P2 könnte sprechen, daß es von der Rohdatenerfassung keine direkte Verbindung zu diesem Paketträger geben würde. Gegen P5 könnte sprechen, daß dieser Prozessor schon durch die Rohdatenerfassung ausgelastet ist. In jedem Fall wird man jedoch auf P3 und P4 einen Paketträger plazieren.

Die graphische Darstellung der Konfiguration mit sechs Prozessoren ist in Abbildung 4.1 gegeben. Formal betrachtet handelt es sich bei der Plazierung der Prozesse um eine qualitative, zusammenhängende, kontraktive, vollständige und konservative Zuordnung (vgl. Abschnitt 3.1.2), die wie oben dargestellt, manuell erfolgt.

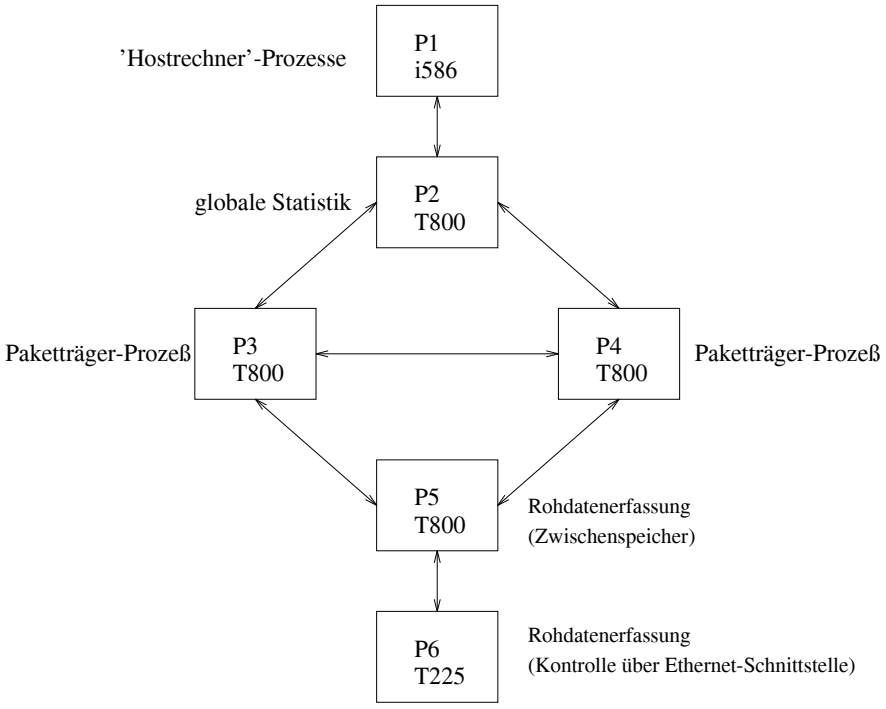


Abbildung 4.1: 6-Prozessor-Konfiguration

4.4 Entwurf der essentiellen Funktionen

Die im letzten Abschnitt vorgestellte Grobeinteilung kann weiter verfeinert werden. Durch die Festlegung der Hardware (Anzahl der Prozessoren und deren Verbindungen untereinander) konnte eine erste Grobgliederung in Abbildung 4.1 vorgestellt werden. Die Plazierung der Prozesse aus Abschnitt 4.3.2.1–4.3.2.4 wurde entsprechend ihrer jeweiligen Hauptaufgabe vorgenommen und im letzten Abschnitt (4.3.3) begründet.

Die Hauptaufgabe der Prozesse — durch Namen wie 'Paketträger', 'globale Statistik' verdeutlicht — wird von Funktionen realisiert. Diese Funktionen sind somit *essentiell* für den jeweiligen Prozeß und für den Prototyp (als paralleles Programm). In den folgenden Abschnitten werden diese essentiellen Funktionen der einzelnen Prozesse entworfen.

4.4.1 Entwurf der essentiellen Funktionen für die Rohdatenerfassung

Die essentiellen Funktionen für die Rohdatenerfassung teilen sich auf zwei Prozesse auf. Einerseits benötigt man einen Prozeß zur physikalischen Ansteuerung und Kontrolle der Ethernet-Schnittstelle. Andererseits benötigt man einen Prozeß für das Zwischenspeichern der erfaßten Daten vor der Weitergabe an die Paketträger. Die essentiellen

Funktionen dieses Prozesses werden in Abschnitt 4.4.5 vorgestellt.

Diese Einteilung in zwei Prozesse ist kontra-intuitiv. Sie ergibt sich aus einer Restriktion der Hardware. Der in Abbildung 3.8 dargestellte Speicher von 64Kbyte reicht gerade aus, um die Funktionen zur Ansteuerung der Ethernet-Schnittstelle aufzunehmen. Die Zwischenspeicherung größerer Datenmengen ist in diesem Speicher nicht möglich. Eine Zwischenspeicherung ist aber erforderlich, um die Rohdatenerfassung von der Analyse zu entkoppeln.

Die Hauptaufgabe der Rohdatenerfassung beschränkt sich demnach auf die Ansteuerung und Kontrolle der Ethernet-Schnittstelle. Die Architektur dieser Schnittstelle wurde bereits in Abschnitt 3.2.1.5 kurz angesprochen. Zur Ansteuerung der eigentlichen Ethernet-Schnittstelle muß auf dem T225-Transputer ein Gerätetreiber geladen werden, der folgende Aufgaben übernimmt:

- Initialisieren der Schnittstelle,
- Ausführen von Funktionstests,
- Konfigurieren der Schnittstelle,
- Festlegen einer Ethernet-Adresse für die Schnittstelle,
- Verwalten von Sende- und Empfangswarteschlange,
- Führen von (netznahen) Statistiken,
- Empfangsmodus wählen,
- Senden von Dateneinheiten,
- Empfangen von Dateneinheiten,
- gegebenenfalls Generieren von Fehlermeldungen.

4.4.2 Entwurf der essentiellen Funktionen für die Analyse

Die parallele Analyse ist der Kern des Meßwerkzeuges. Mit Hilfe der Analyseergebnisse können Aussagen über die Schnittstellen gemacht werden, an denen die Rohdaten erhoben werden. In Abschnitt 4.4.2.1 wird dieser Zusammenhang dargestellt.

Ein wichtiges Konzept beim Auswerten der Rohdaten ist die Unterscheidung zwischen *Erkennen* und *Analysieren* der Dateneinheiten. Dieses Konzept wird in Abschnitt 4.4.2.2 vorgestellt, da bei der Realisierung der einzelnen Analysefunktionen ebenfalls zwischen Erkennen und Analysieren unterschieden werden soll.

In Abschnitt 4.4.2.3 wird eine allgemeine Analysefunktion entworfen.

4.4.2.1 Schnittstellen bei der Datenerfassung

Zu Beginn muß auf die *Schnittstellen* eingegangen werden, an der die Meßdaten erhoben werden. Nach der Datenerhebung an diesen Schnittstellen können durch die

Analyse Aussagen über die betrachtete Schnittstelle gemacht werden. Sind die Schnittstellen definiert, kann man die jeweiligen Analysefunktionen für die einzelnen Schnittstellen diskutieren. Bei der Realisierung dieser Funktionen muß vor allem auf die Erweiterbarkeit geachtet werden. Hierzu mehr in Abschnitt 5.3.2.

Offensichtlich ist die einzige physikalische Schnittstelle des Meßwerkzeuges zum Netz die Ethernet-Schnittstelle. Man kann nun *Systemmessungen* durchführen. Das System ist in diesem Fall das Netz. An dieser Schnittstelle können die verschiedenen *Aufträge* und deren *Parameter* gemessen werden. Ein möglicher Auftrag wäre das Versenden eines Ethernet-'Frames'. Mögliche Parameter wären beispielsweise die Länge und die Prüfsumme.

Neben diesen reinen Systemmessungen kann man auch *Lastmessungen* an einem System durchführen. Dazu untersucht man beispielsweise die verschiedenen Aufträge und die Zwischenankunftszeiten, mit denen diese Aufträge an das System abgegeben werden. Aufträge könnten in diesem Fall Ethernet-'Frames' sein, die an das Netz zur Übertragung abgegeben werden. Alle Dateneinheiten, die in einem bestimmten Zeitintervall an das System abgegeben werden, bilden die Last (vgl. [WK90]).

Eine dritte Möglichkeit sind *Leistungsmessungen*. Dazu werden mit Hilfe von *Lastgeneratoren* Aufträge oder Auftragsklassen an das System übergeben und beobachtet, wie sich das System unter dieser Last verhält.

Eine Besonderheit ergibt sich bei Messungen an Ethernet-basierten Kommunikationssystemen. Durch die oben besprochenen Filter kann man die Schnittstelle derart einschränken, daß nur noch ein Teil des Kommunikationssystems (ein Subsystem) betrachtet wird. Beispielsweise könnte man anstelle aller Aufträge, die an das Netz abgegeben werden, nur noch die Aufträge messen, die von einem bestimmten Knoten an das Netz abgegeben werden.

Außerdem kann man aufgrund der hierarchischen Strukturierung der Dienste in einem Kommunikationssystem unter Nutzung einer Hierarchie von Protokollen, jede Protokollschicht als Schnittstelle für eine bestimmte Klasse von Aufträgen ansehen. Diese Schnittstellen sind durch die Protokolleinbettung[7] *indirekt* zugänglich für Messungen (vgl. Abbildung 4.2).

[7] Es werden gegebenenfalls Dateneinheiten einer Protokollschicht in Dateneinheiten einer anderen Protokollschicht eingebettet. Entsprechend werden Dateneinheiten eines Protokolls erkannt. Im folgenden wird dies durch 'Protokolleinbettung' und 'Erkennen von Protokollen' abgekürzt.

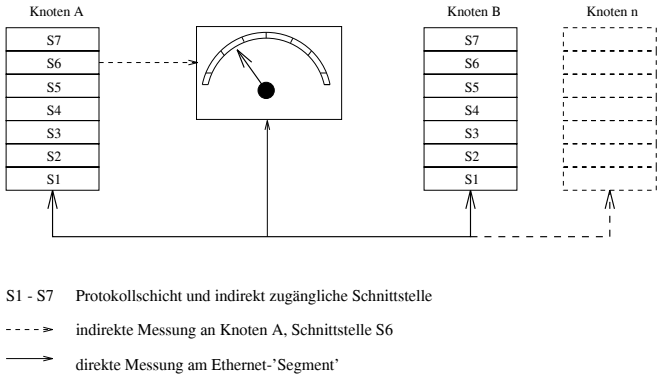


Abbildung 4.2: Direkte und indirekte Messung

Zusammenfassend ergibt sich, daß jede Protokollschicht im Kommunikationssystem als Schnittstelle für Messungen zugänglich sein kann. Darüber hinaus kann die Messung auf einen Teil bzw. auf Teile des Kommunikationssystems eingeschränkt werden. Beispielsweise könnte man nur Aufträge eines Knotens im Netz messen oder nur Aufträge an einer bestimmten Schnittstelle, die durch Dateneinheiten eines bestimmten Protokolls erkennbar ist.

4.4.2.2 Erkennen und Analysieren von Protokollen

Aufgrund des überwiegend indirekten Zugriffs des Meßwerkzeuges auf eine Schnittstelle muß man beim Auswerten der Meßdaten in zwei Schritten vorgehen. Für jede Dateneinheit, die über die Ethernet-Schnittstelle empfangen wird, muß festgestellt werden, ob sie auch Informationen über eine Schnittstelle beinhaltet, für die Meßdaten analysiert werden sollen. Dieser erste obligatorische Schritt bei der Auswertung wird im folgenden als *Erkennen* bezeichnet. Da die zu erkennenden Schnittstellen in diesem Fall die Schnittstellen zwischen den Protokollebenen sind, muß erkannt werden, ob in dem jeweiligen Ethernet-'Frame' Dateneinheiten der zu analysierenden Protokolle enthalten sind. Der zweite Schritt ist die *Analyse*, unter der hier das Selektieren und Berechnen von Informationen auf — Grundlage von Benutzerspezifikationen — aus den erkannten Dateneinheiten verstanden wird.

Erkennen von Dateneinheiten

Ziel der Erkennung von unausgewerteten Dateneinheiten ist eine Typisierung. Die Typisierung muß, aufgrund der Einbettung von Dateneinheiten eines Protokolls in Dateneinheiten eines tieferliegenden Protokolls, innerhalb der Protokollhierarchie iterativ erfolgen. Ist der Typ einer Dateneinheit auf der Protokollebene (n) erkannt worden, kann eine weitere Typisierung auf der Ebene (n+1) erfolgen, bis schließlich die gesamte Sequenz der sukzessive angewendeten Protokolle für die Dateneinheit bekannt ist oder keine weitere Typisierung möglich ist. Eine weitere Typisierung ist

genau dann unmöglich, wenn auf einer Protokollebene ein unbekanntes Protokoll verwendet wurde. Die fehlende Kenntnis über das PDU-Format der Dateneinheiten des unbekannten Protokolls verhindert die Selektion der für die weitere Identifizierung notwendigen Informationen aus der Dateneinheit.

Das Ergebnis der Erkennung ist eine teilweise oder vollständige Angabe der verwendeten Protokollhierarchie. Beispielsweise könnte eine Dateneinheit folgende Typisierung haben: (Ethernet-'Frame'/IP/TCP/TELNET). Hierbei handelt es sich um eine vollständige Typisierung, da kein weiteres Protokoll in TELNET eingebettet wird. Im ersten Iterationsschritt mußte der Ethernet-'Frame' aus den beiden Möglichkeiten (Ethernet-'Frame', 'IEEE 802.3 Frame') identifiziert werden. Im zweiten Iterationsschritt mußte das Internetprotokoll (IP) aus einer Vielzahl möglicher Protokolle identifiziert werden. Ergebnis der dritten und vierten Iteration war die Identifizierung des 'Transmission Control Protocol' (TCP) und die Identifizierung von TELNET.

In Abbildung 4.3 ist dargestellt, welche Informationen[8] für das Erkennen einer Dateneinheit notwendig sind. Der dargestellte Baum ist bei weitem nicht vollständig. Es soll nur verdeutlicht werden, welche Informationen einer Dateneinheit das iterative Erkennen der verwendeten Protokollhierarchie ermöglichen. Gerade diese Unterschiede in den auszuwertenden Informationen auf den verschiedenen Protokollebenen zwingen zu einem iterativen Erkennen der Protokollhierarchie. Hier wird noch einmal die Nähe der Messungen am Kommunikationssystem zur Protokollverarbeitung in Endsystemen deutlich (vgl. Abschnitt 3.3.2).

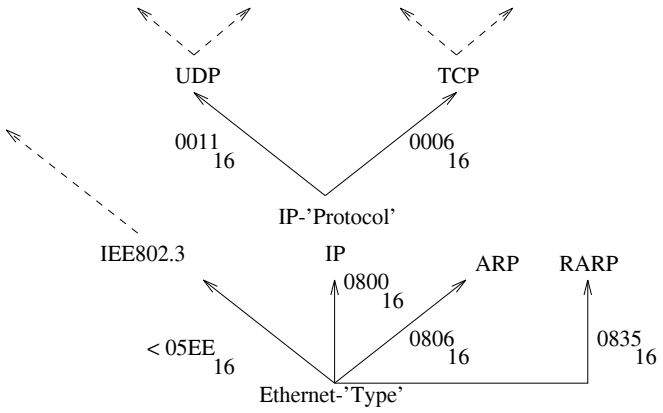


Abbildung 4.3: Erkennungsbaum für Protokolle

[8] Ethernet-'Type' und IP-'Protocol' sind Beispiele für solche Informationen. Es handelt sich hierbei um Werte, die aus der Dateneinheit des jeweiligen Protokolls gewonnen werden (vgl. Abschnitte 5.3.2.2–5.3.2.7).

Analysieren von erkannten (typisierten) Dateneinheiten

Wurde im ersten Schritt eine Dateneinheit erkannt, kann diese Dateneinheit analysiert werden. Eine Analyse ist bereits nach einer partiellen Erkennung möglich. So kann eine als Ethernet-'Frame' typisierte Dateneinheit bereits analysiert werden, ohne daß zuvor in weiteren Iterationen eine genauere Typisierung vorgenommen wird. Voraussetzung für eine Analyse auf der Protokollebene (n) ist demnach nur eine Identifizierung (Erkennung) bis zur Protokollebene (n). Das Ergebnis der Analyse ist die vollständige Beschreibung der variablen Informationen einer Dateneinheit und die Verifikation der invarianten Informationen einer Dateneinheit auf einer bestimmten Protokollebene. Damit beinhaltet die Analyse einer Dateneinheit der Ebene (n) auch das Erkennen des Typs einer eingebetteten Dateneinheit der Ebene (n+1), sofern vorhanden. Die vollständige Beschreibung einer Dateneinheit ist protokollspezifisch. Eine Analyse kann demnach nur durchgeführt werden, wenn das zuvor erkannte Protokoll dem Analysewerkzeug auch bekannt ist.

Typischerweise kann ein Analysewerkzeug wesentlich mehr Protokolle erkennen als analysieren. Dies liegt vor allem an der deutlich geringeren Komplexität des Erkennens im Vergleich zur Analyse. Die vollständige Beschreibung einer Dateneinheit beinhaltet neben protokollspezifischen Angaben weitere Angaben. Beispielsweise hat jede Dateneinheit eine bestimmte Länge in Byte, die für jede Dateneinheit verschieden sein kann. Längenangaben können wiederum zu weiteren Informationen führen. Beispielsweise kann über die Längenangaben aller Dateneinheiten innerhalb eines bestimmten Zeitintervalls eine (Brutto-)Last berechnet werden. Für ein bestimmtes Analysewerkzeug muß entsprechend den Benutzeranforderungen festgelegt werden, was zu einer vollständigen Beschreibung einer Dateneinheit gehört.

4.4.2.3 Anforderungen an die Analysefunktionen

Die zu realisierenden Analysefunktionen müssen für jedes zu analysierende Protokoll entwickelt werden. Entscheidend dabei ist, daß sie so flexibel miteinander kombinierbar sein müssen, daß bei späteren Erweiterungen möglichst keine bzw. nur minimale Änderungen an existierenden Funktionen vorzunehmen sind. Nur so sind einfache Erweiterungen möglich. Die einzelnen Analysefunktionen sollten folgendes leisten:

- Erkennen eingebetteter Dateneinheiten und Identifizieren des verwendeten Protokolls
- Analysieren der Dateneinheit
- Speichern der Analyseergebnisse in einem einheitlichen Format
- Aufrufen der Analysefunktion für eine erkannte, eingebettete Dateneinheit
- Berechnen lokaler Statistiken zur Unterstützung des globalen Statistikprozesses

4.4.3 Entwurf der essentiellen Funktionen für die globale Statistik

Bei dem Konzept der Paketparallelität fallen Analysedaten typischerweise[9] dezentral (bei den verschiedenen Paketträgerprozessen) an. Zwar kann jeder Paketträger lokale Statistiken (beispielsweise zur Anzahl der empfangenen Dateneinheiten eines bestimmten Typs) führen, die globale Statistik muß jedoch von einem Prozeß übernommen werden, der Zugriff auf die Analyseergebnisse aller Paketträger hat (z.B. Gesamtanzahl der empfangenen Dateneinheiten eines Typs).

Im allgemeinen Fall, bei einer großen Anzahl von Paketträgerprozessen, kann es auch notwendig werden, mehrere Prozesse für die globale Statistik vorzusehen. Die Prozesse könnten hierarchisch angeordnet werden. Auf der untersten Hierarchiestufe werden die Analyseergebnisse von den Paketträgern weiterverarbeitet. Auf höheren Hierarchiestufen werden diese Zwischenergebnisse erneut gebündelt.

Der Prototyp benötigt einen Prozeß für die globale Statistik. Die einzige Aufgabe der globalen Statistik besteht darin, die 'Standardauswertung' (vgl. Abschnitt 2.7.4) vorzubereiten. Als Standardauswertung soll die Auslastung des Ethernet-Segmentes periodisch angezeigt werden. Die Auslastung soll entsprechend den verwendeten Protokollen aufgeschlüsselt werden.

Die globale Statistik muß demnach Funktionen bereitstellen, die

- periodisch (z.B. jede Sekunde) lokale Statistiken der Paketträger sammeln,
- die Auslastung entsprechend der verwendeten Protokolle berechnen und
- die Ergebnisse in ein Format umwandeln, daß dem 'Hostrechner' eine schnelle Weiterverarbeitung (z.B. graphische Präsentation) der Ergebnisse ermöglicht.

Sollen zusätzlich zu dieser Standardauswertung weitere Statistiken bereitgestellt werden, kann nach dem gleichen Schema vorgegangen werden. Entscheidend ist, daß die globale Statistik von Funktionen zur lokalen Statistik in den Paketträgern unterstützt wird (s.o.). Der globale Statistikprozeß könnte zum Engpaß im Prototypen werden, wenn er alle Rohdaten selber statistisch auszuwerten hätte.

4.4.4 Entwurf der essentiellen Funktionen für den 'Hostrechner'

Aus der Darstellung in Abschnitt 4.3.2.4 geht hervor, daß der 'Hostrechner' (Host) viele verschiedene Funktionen bereitstellen muß, um die

- Interaktion mit dem Benutzer zu ermöglichen, die
- Analyseergebnisse darzustellen und zu exportieren und die
- Konfigurierung der Transputer vorzunehmen.

[9] Ausnahmen bilden Konfigurationen mit nur einem Paketträgerprozeß. In diesem Fall handelt es sich zwar immer noch um Parallelverarbeitung, da noch andere Prozesse nebenläufig zum Paketträgerprozeß ablaufen, jedoch nicht mehr um Paketparallelität im engeren Sinne. Die eigentliche Analyse oder auch Protokollverarbeitung wäre dann sequentiell!

4.4.4.1 Essentielle Funktionen für die Interaktion mit dem Benutzer

Die Steuerung des Prototypen wird vom Benutzer übernommen. Dazu muß der Benutzer Aktionen im Meßwerkzeug auslösen können. Bei der Verwendung einer graphischen Benutzeroberfläche gibt es prinzipiell die beiden folgenden Möglichkeiten:

- Der Benutzer kann Aktionen über Kommandos auslösen, die über eine *Kommandozeile* eingegeben werden. Der Benutzer tippt dazu über die Tastatur die entsprechenden Kommandos ein.
- Der Benutzer kann Aktionen durch Auswahl graphischer Symbole[10] mit der 'Maus' auslösen.

In Abschnitt 4.3.2.4 wurde bereits dargestellt, daß beide Eingabemöglichkeiten Vorteile haben. Es sollen daher Funktionen für eine Kommandozeile und Funktionen zur Eingabe über 'Dialoge' und 'Menüs' realisiert werden.

Die Kommandozeile muß

- Eingaben über die Tastatur darstellen,
- komfortable Editierfunktionen bieten (Einfügen, Löschen, Verschieben und Überschreiben von Zeichen(ketten)) und
- die der Eingabe entsprechenden Aktionen auslösen können.

Die 'Dialoge' und 'Menüs' sollten in ihrer Erscheinung den unter OS/2 üblichen Dialogfeldern und Menüstrukturen ähnlich sein, um dem Benutzer eine intuitive Auswahl zu ermöglichen (vgl. Abschnitt 2.6)

Die Ausgabe von Fehlermeldungen und textbasierter Informationen soll in einem 'Fenster' der graphischen Oberfläche erscheinen. Dazu werden Funktionen benötigt, die

- die Informationen in das Fenster einschreiben,
- Größe und Lage des Fensters auf der graphischen Oberfläche kontrollieren,
- optional die Ausgaben zusätzlich in eine Datei schreiben und
- aus dem Fenster 'herausgerollte' Informationen durch Verschieben des Fensterausschnittes[11] erneut anzeigen.

[10] Symbole können Elemente eines Menüs oder Schalter und ähnliche Kontrollelemente eines 'Dialogfeldes' sein. Die Terminologie bei der Programmierung der graphischen Schnittstelle von OS/2 wird beispielsweise in [Eck93] dargestellt.

[11] Die grundlegenden Techniken der 'Fensterprogrammierung' werden beispielsweise in [Roc90] dargestellt.

4.4.4.2 Essentielle Funktionen zur Darstellung und Export der Analyseergebnisse

Analysedaten werden entweder 'on-line' dargestellt oder in einer Datei gespeichert. Werden die Daten 'on-line' angezeigt, kann eine beliebige programminhärente Darstellung gewählt werden. Sollen hingegen externe Werkzeuge auf den Daten arbeiten, muß man eine möglichst kompakte, maschinell leicht weiterverarbeitbare Darstellung wählen (vgl. Abschnitt 2.5 und 2.7.1).

Soll die Datei nachträglich von einem (menschlichen) Benutzer analysiert werden, muß ein drittes Datenformat gewählt werden. Der Informationsgehalt ist u.U. derselbe wie bei der maschinennahen Darstellung. Jedoch müssen für eine gute Lesbarkeit der Daten zusätzliche redundante Informationen mit in die Datei aufgenommen werden.

Für die programminterne Darstellung von Analysedaten sollen weitere 'Fenster' der graphischen Benutzeroberfläche verwendet werden. Die Ausgabe kann entweder als Textmeldung erfolgen (analog zu Fehlermeldungen (s.o.)) oder alternativ als Graphik. Insbesondere die 'Standardauswertung' (Auslastung des Ethernet-Segmentes) soll als Liniengraphik dargestellt werden (vgl. Abschnitt 2.4 und 2.7.4).

4.4.4.3 Essentielle Funktionen zur Konfigurierung der Transputer

Die Transputer sollen beim Programmstart vom Host automatisch konfiguriert werden (vgl. Abschnitt 4.3.2.4). Dazu sind folgende Funktionen notwendig:

- Initialisieren der Transputer. Zu jedem Zeitpunkt muß es möglich sein, die Transputer in einen definierten Zustand zurückzusetzen. Beispielsweise muß vor dem Laden von Programmen auf einen Transputer (s.u.) sichergestellt werden, daß der Transputer in einem bestimmten Betriebsmodus ('Bootmode') ist. Um einen 'Reset' an einem Transputer durchzuführen, ist es notwendig, eine Verbindung zu dem entsprechenden Transputer herzustellen. Eine solche Verbindung muß beim Einsatz der TBs aus Abbildung 3.7 erst programmiert werden (s.u.).

- Programmieren der Topologie. Verbindungen zwischen 'Transputerlinks' werden durch Programmieren des Kreuzschienenverteilers (Abbildung 3.7) hergestellt. Eine Verbindung wird dabei durch Einschreiben eines Tupels $K = (k_1, k_2)$ in ein Register des Kreuzschienenverteilers hergestellt. k_1 und k_2 identifizieren 'Transputerlinks' oder externe Verbindungsbuchsen mit eindeutigen numerischen Werten (vgl. Abschnitt 3.2.1.5). Eine Funktion zur Programmierung der Topologie sollte in jedem Fall verhindern, daß unzulässige Verbindungen hergestellt werden. Eine Bezeichnung der 'Transputerlinks' mit Namen (symbolische Bezeichner) anstelle der numerischen Werte würde außerdem die Benutzung erleichtern.

- Laden der Transputerprozesse. Prozesse, die in ein Transputernetz geladen werden sollen, werden von der Entwicklungs-Software automatisch mit Ladefunktionen ergänzt, die für die Verteilung der Prozesse auf die einzelnen Prozessoren zuständig sind. Die Verteilung muß vom Benutzer in einer Datei (Konfigurationsskript) spezifiziert werden. Zum Laden der Transputerprozesse wird auf dem

Host nur eine Funktion benötigt, die das parallele Programm (Transputerprozesse mit Ladefunktionen) an den Transputer sendet, der an die Schnittstelle zum Host angeschlossen ist. Zuvor müssen die Transputer, wie oben dargestellt, initialisiert werden.

4.4.5 Entwurf essentieller Funktionen zum Filtern

Die Funktionen zum Filtern der Rohdaten sind bisher noch keinem Prozeß zugeordnet worden. In Abschnitt 4.4.5.1 wird darauf eingegangen. Unabhängig davon, wann und wo gefiltert wird, müssen die Filterfunktionen entworfen werden. Der Entwurf der Filterfunktionen wird in Abschnitt 4.4.5.2 dargestellt.

4.4.5.1 Zeitpunkt des Filterns

Die weiter oben diskutierten Funktionen konnten bereits Prozessen zugeordnet werden. Funktionen zum Filtern der Rohdaten sind ebenfalls unbedingt erforderlich (vgl. Abschnitt 2.7.4). Die Zuordnung zu einem Prozeß kann an dieser Stelle jedoch noch nicht entschieden werden. Das Filtern sollte so früh wie möglich geschehen, um die Last (durch Rohdaten) im Werkzeug zu verringern (vgl. Abschnitt 1.3.2.1). Insbesondere vor Ausführen der Analysefunktionen ist eine Reduktion der Rohdaten wünschenswert. Die beiden folgenden Möglichkeiten zum Ausführen der Filterfunktionen sind denkbar:

1. Filtern im Paketträger. Die Filterfunktionen werden unmittelbar vor dem Aufruf der Analysefunktionen ausgeführt.

2. Filtern im Prozeß zur Rohdatenerfassung. Unmittelbar nach der Rohdatenerfassung werden die Filterfunktionen aufgerufen.

Das Filtern im Paketträger hat den Vorteil, analog zur Paketparallelität ein paralleles Filtern der Dateneinheiten zu ermöglichen. Es gäbe in diesem Fall eine der Anzahl der Paketträger entsprechende Anzahl an Filterfunktionen. Dieses Vorgehen ist angebracht, wenn das Filtern einer Dateneinheit aufwendig und zeitintensiv ist. Ein einzelner Prozeß könnte durch das Filtern zum Engpaß werden.

Bei Messungen in Verteilten Systemen ist die erste Alternative ebenfalls vorzuziehen. In einem Verteilten System kann es vorkommen, daß mehrere Anwendungen auf die Meßdaten zugreifen, so daß es bei der Rohdatenerfassung nicht möglich ist zu entscheiden, welche Daten gefiltert werden können.

Ein Vorteil der zweiten Alternative besteht in der Möglichkeit, die Filterfunktionen in einem eigenen Prozeß zu kapseln. Das erhöht die Modularität des Werkzeuges. Außerdem vermeidet ein Filtern unmittelbar nach der Rohdatenerfassung das Versenden von nicht benötigten Rohdaten an die Paketträger.

Unabhängig von dieser Reduktion der Rohdaten können weitere Filtervorgänge zu einem späteren Zeitpunkt erforderlich sein. Beispielsweise könnte sich im Zuge der Analyse durch die Paketträger ergeben, daß bestimmte Informationen der Rohdaten

nicht benötigt werden. Diese Informationen könnten in einer zweiten 'Filterphase' (beispielsweise durch den globalen Statistikprozeß) vernichtet werden.

4.4.5.2 Entwurf der Filterfunktionen

Auch bei der Definition und Eingabe von Filtern gibt es verschiedene Möglichkeiten:

1. Vorgabe von Standardfilter[12]. Der Benutzer kann in diesem Fall aus einer vorgegebenen Menge von Standardfilter auswählen.
2. Universelle Filter. Der Benutzer spezifiziert beliebige Bitmuster, die an bestimmten Positionen in den Dateneinheiten auftreten müssen.
3. Eine Kombination aus universellen Filtern und Standardfiltern.

Ein Vorteil der Standardfilter ist, daß der Benutzer nur auszuwählen braucht, *welche* Dateneinheiten weiterverarbeitet werden sollen, jedoch nicht, *wie* gefiltert werden soll[13]. Außerdem können Standardfilter über 'Dialogfelder' einer graphischen Benutzeroberfläche (s.o.) ausgewählt werden. Ein Nachteil dieser Lösung ist, daß Standardfilter 'a priori' bei der Realisierung vorgegeben werden müssen. Es können dann nur Filter ausgewählt werden, die der Entwickler bei der Realisierung als notwendig ansieht.

Flexibler als Standardfilter sind universelle Filter. Durch diese Filter können beliebige Dateneinheiten selektiert oder vernichtet werden. Es sind auch keine Änderungen oder nachträgliche Erweiterungen erforderlich. Der Nachteil dieser Lösung ist, daß der Benutzer über das Expertenwissen verfügen muß, um spezifizieren zu können, wie der Filtervorgang durchzuführen ist. Der Benutzer muß selber die Kriterien angeben, die für die Entscheidung notwendig sind, welche Dateneinheiten zu vernichten sind.

Alternativ kann man die Vorteile beider Verfahren kombinieren. Der Benutzer wählt weitgehend passende Standardfilter aus, die in universelle Filter übersetzt werden. Die universellen Filter können dann vom Benutzer verändert oder erweitert werden.

Da Standardfilter zu inflexibel sind und eine Realisierung der Kombination beider Filtertechniken aus Zeitgründen nicht möglich ist, soll der Prototyp mit universellen Filtern realisiert werden.

Eine weitere Schwierigkeit beim Entwurf dieser Funktionen liegt darin, daß die Analyse und damit das Filtern der Rohdaten auf einem anderen Prozessor abläuft, als die Definition der Filter. Ein weiteres Problem in diesem Zusammenhang ist, daß beim Filtern nicht nur einfache, direkt kodierbare Vergleiche durchzuführen sind, sondern über Präzedenzrelationen anzuordnende Kombinationen von Vergleichsoperationen. Es muß demnach Codegenerierung für das Filtern zur Laufzeit des Analysewerkzeuges möglich sein.

[12] Beispiele für Standardfilter: 'Selektiere alle IP-Dateneinheiten', 'Selektiere die Dateneinheiten des Knotens X.'

[13] Der Benutzer benötigt demnach kein Wissen über den Aufbau von Dateneinheiten und Knotenadressen.

Da die Definition von Filtern auf einem anderen Prozessor[14] erfolgt als das Filtern, muß entschieden werden, ob die Filterdefinition oder bereits generierter Programmcode an den anderen Prozessor gesendet werden soll. Außerdem muß entschieden werden, ob Programmcode für einen abstrakten Prozessor (sogenannter Zwischencode oder 'P-Code') oder Maschinencode für die Transputer erstellt wird.

Da bei Fehlern in der Filterdefinition der 'Hostrechner' ohnehin für die Ausgabe entsprechender Fehlermeldungen zuständig ist, sollen Filterdefinitionen auch auf dem 'Hostrechner' in Programmcode übersetzt werden. Eine weitere wichtige Begründung für diese Entscheidung ist, daß sowohl die syntaktische als auch die semantische Überprüfung der Filterdefinitionen von Funktionen übernommen werden können, die auch die Überprüfung der Eingaben in der Kommandozeile (vgl. Abschnitt 4.4.4.1) durchführen. Diese Funktionen können so in einem Modul für den 'Hostrechner' zusammengefaßt werden.

Außerdem wurde entschieden, Zwischencode für einen abstrakten Prozessor zu generieren, da dies weniger aufwendig erscheint als die direkte Maschinencodegenerierung für Transputer. Ein Nachteil von 'P-Code' ist der langsamere Ablauf, da dieser 'P-Code' durch den abstrakten Prozessor interpretiert werden muß.

4.5 Entwurf der unterstützenden Funktionen

Die unterstützenden Funktionen stellen Dienste für die essentiellen Funktionen zur Verfügung. Alle unterstützenden Funktionen könnten alternativ in die essentiellen Funktionen integriert werden. Die Programmquellen sollten jedoch möglichst modular erstellt werden. Durch eine Aufteilung in essentielle Funktionen und Funktionen zur Unterstützung sind Erweiterungen leichter möglich. Weitere Gründe für diese Einteilung werden in den folgenden Abschnitten genannt.

4.5.1 Entwurf von Vermittlungsfunktionen und Verteilungsfunktionen

Im allgemeinen Fall gibt es mehrere Statistikprozesse, mehrere Paketträgerprozesse und eventuell auch mehrere Schnittstellen zu einem Netz oder zu verschiedenen Netzen mit den dafür nötigen Prozessen für die Ansteuerung der Schnittstellen (Rohdatenerfassung). Vor diesem Hintergrund ist zu überlegen, wie die verschiedenen Prozesse beim Nachrichtenaustausch adressiert werden können.

Die einfachste Lösung besteht darin, die Prozesse einer Konfiguration zu numerieren. Für jeden Prozessor und jeden Prozeß der Konfiguration kann dann ein Weg für eine Nachricht von Sendeprozeß zu Empfangsprozeß angegeben werden. Diese Vorgehensweise ist offensichtlich sehr unflexibel, da

[14] Die Definition von Filtern erfolgt über die Kommandozeile und somit auf dem 'Hostrechner'. Dieser verfügt über einen INTEL-Prozessor und nicht über einen Transputer. Es handelt sich also sowohl um einen anderen als auch um einen andersartigen Prozessor.

- jede Veränderung der Konfiguration eine Neuberechnung der Nachrichtenwege impliziert,
- keine Prozeßgruppen angesprochen werden können[15] und
- jeder Prozeß, der Nachrichten versenden möchte, Kenntnis von allen potentiellen Empfängern seiner Nachricht haben muß.

Wesentlich komfortabler, übersichtlicher und leicht an neue Konfigurationen anpaßbar wäre eine Lösung, bei der die Nachrichten auf jedem Prozessor weitervermittelt werden. Außerdem wird man Namen für Prozeßgruppen vergeben wollen, um nicht jeden Prozeß einzeln adressieren zu müssen. Eine Nachricht wird nur dann mehrfach versendet, wenn die Empfänger über verschiedene physikalische Verbindungen ('Transputerlinks') zu erreichen sind. Die Kenntnis über Prozeßgruppen und die Entscheidung, ob eine Nachricht vervielfältigt werden muß, sollte in Vermittlungsfunktionen verborgen werden. Der Nachrichtenaustausch wäre für die Prozesse transparent. Auch der (obligatorische) gegenseitige Ausschluß beim Zugriff auf die 'Transputerlinks' kann in den Vermittlungsfunktionen verborgen werden.

Ein weiterer Grund, die Nachrichtenvermittlung als unterstützende Funktionen bereitzustellen, ist die Erfahrung, daß die Trennung von Verarbeitung und Kommunikation in der Regel die Verarbeitungsleistung der Transputer erhöht (vgl. [Ben94, S.114f]).

Darüber hinaus sollten die Vermittlungsfunktionen weiter unterteilt werden in Funktionen zum Senden und Funktionen zum Empfangen. Realisiert man die Sende- und Empfangsfunktionen als eigenständige Prozesse, vermeidet man Verklemmungen, wie in Abbildung 4.4 dargestellt.

[15] Eine Nachricht, die bei einer solchen Lösung an mehr als einen Prozeß gesendet werden soll, muß für alle Empfangsprozesse einzeln versendet werden. Auch das 'Benennen' von Prozeßgruppen mit eindeutigen Nummern ändert an der Notwendigkeit, Nachrichten mehrfach zu versenden, nichts. Vielmehr muß jeder Prozeß neben den einzelnen Nummern aller anderen Prozesse auch deren Zugehörigkeit zu den verschiedenen Gruppen kennen.

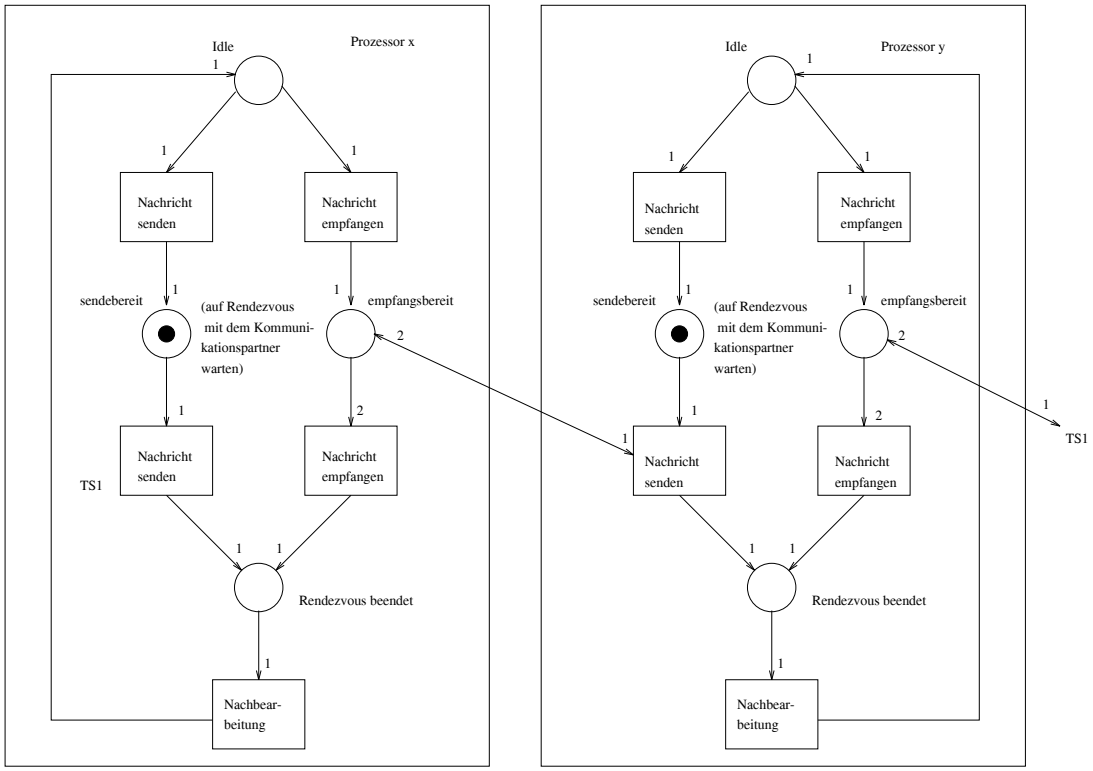


Abbildung 4.4: Verklemmung beim Nachrichtenaustausch

Die Vermittlungsfunktionen sollen folgende Funktionalität umfassen:

- Empfangen von Nachrichten über auswählbare Kommunikationskanäle. Ein Kommunikationskanal ist dabei entweder einer der Transputerlinks (vgl. Abschnitt 3.2.1.4) oder ein interner Kanal eines anderen Prozesses.
- Vermitteln von Nachrichten über die ausgewählten Kommunikationskanäle gemäß einer Vermittlungstabelle.
- Unterstützen von 'Multicast- und Broadcast'-Nachrichten.
- Entkoppeln der Sendefunktionen von den Empfangsfunktionen zur Vermeidung von Verklemmungen beim Nachrichtenaustausch (vgl. Abbildung 4.4).

Darauf aufbauen können spezielle Verteilungsfunktionen. Beispielsweise muß es eine solche Verteilungsfunktion geben, um die Dateneinheiten auf die Paketträgerprozesse zu verteilen. Verteilungsfunktionen können analog zu den Funktionen für die globale Statistik (s.o.) hierarchisch angeordnet werden, falls nicht alle Paketträgerprozesse direkt benachrichtigt werden können. Bei der 6-Prozessor-Konfiguration gibt es nur zwei gleichartige essentielle Prozesse, die Paketträger. Nur für diese beiden Pro-

zesse benötigt man eine Verteilungsfunktion, die die Dateneinheiten auf sie aufteilt. Die Dateneinheiten könnten mit einer Verteilungsfunktion

- abwechselnd oder
- zufällig oder
- abhängig von der Auslastung der Paketträger

verteilt werden. Es sind weitere Strategien denkbar, wie man die Dateneinheiten auf die Paketträger aufteilen könnte. Der Prototyp soll mit der einfachsten Strategie realisiert werden. Die Dateneinheiten werden abwechselnd auf die Paketträger verteilt.

4.5.2 Entwurf von Verbindungsmanagement-Funktionen

Die Verbindungsmanagement-Funktionen werden nur dann benötigt, wenn bei der Analyse von verbindungsorientierten Protokollen Zustandsänderungen der Protokollautomaten beobachtet werden sollen oder das jeweilige Protokoll getestet werden soll.

Die Verbindungsmanagement-Funktionen müßten in diesem Fall:

- *Verbindungen erkennen.* Eine Messung kann zu einem beliebigen Zeitpunkt gestartet werden. Zu diesem Zeitpunkt kann es bereits etablierte Verbindungen geben, Verbindungen können gerade aufgebaut oder auch beendet werden.

  Treffen Dateneinheiten von verbindungsorientierten Protokollen ein, müssen die Verbindungen und deren aktueller Zustand[16] erkannt werden. Dies ist die Grundlage für alle weiteren Funktionen dieser Kategorie (s.u.). Das Erkennen einer Verbindung ist meist durch Auswerten mehrerer Informationen der entsprechenden Dateneinheit möglich. Beispielsweise identifiziert das Tupel T = (Senderadresse, Empfängeradresse, Quellport und Senkenport) eine Verbindung bei dem verbindungsorientierten Protokoll TCP[17]. Darüber hinaus kann über andere Informationen aus der Dateneinheit der Zustand der Verbindung rekonstruiert werden.

- *Verbindungen verwalten.* Hierunter fallen Funktionen zum Aktualisieren von Verbindungskontexten und Funktionen zum Zugriff auf Verbindungskontexte (z.B. Streufunktionen[18] über obiges Tupel T).

- *Dateneinheiten 'weiterleiten'.* Diese Funktionen sind notwendig, um Dateneinheiten einer Verbindung, die nicht durch den lokalen Paketträgerprozeß verwaltet werden, an einen anderen Paketträgerprozeß weiterzuleiten. Dies ist notwendig, da Dateneinheiten einer Verbindung nicht automatisch zum 'richtigen' Paketträgerprozeß gelangen, sondern erst ein Paketträger durch eine entsprechende

[16] Beispiele für einen solchen Zustand sind: 'Empfangsbereit', 'Sendebereit', 'Verbindung etabliert' oder auch 'Idle'.

[17] Die Abkürzung steht für das 'Transmission Control Protocol' aus der TCP/IP-Protokollfamilie.

[18] Streufunktionen ('Hash'-Funktionen) ermöglichen einen schnellen Zugriff auf unsortierte Listenelemente (hier: Verbindungskontexte). Ein Vergleich verschiedener Streufunktionen wird beispielsweise in [ASU88, S. 532ff] gegeben. Detaillierte Realisierungsvorschläge findet man in [Hol90, S. 715ff].

Analyse der Dateneinheit entscheiden kann, welcher Prozeß die Verbindung verwaltet.

Die Verbindungsmanagement-Funktionen gehören in die Kategorie der Unterstützungsfunktionen. In diesem Fall muß die parallele Analyse unterstützt werden, um Analyseergebnisse berechnen zu können, die in direktem Zusammenhang mit einer Verbindung durch verbindungsorientierte Protokolle stehen[19].

Beim Prototyp wird auf die Realisierung der Verbindungsmanagement-Funktionen verzichtet. Falls sich das Paradigma der Paketparallelität bei der Analyse bewährt, kann bei Bedarf die Realisierung dieser Funktionen nachgeholt werden.

4.5.3 Entwurf der Kommunikationsunterstützung für die 'Hostrechner'-Transputer-Kommunikation

Die Kommunikation der Transputer untereinander wird durch die 'Transputerlinks' abgewickelt. Die Ansteuerung der 'Transputerlinks' ist trivial. Die Unterstützung der Interprozeß- und der Interprozessor-Kommunikation geht bis auf die Maschinensprachen-Ebene der Transputer. Komplexe Kommunikations- und Kooperationsmechanismen sind dadurch leicht realisierbar. Darüber hinaus müssen die Transputer auch mit dem 'Hostrechner' während der Ausführung des Programms kommunizieren können.

Das Betriebssystem des 'Hostrechners' muß dafür um Funktionen zur Ansteuerung der TBs erweitert werden[20]. Funktionen zur Ansteuerung bestimmter Hardware werden als *Gerätetreiber* (engl. 'Device Driver') bezeichnet. Neben der Ansteuerung der Geräte sollen die Gerätetreiber darüber hinaus von den Besonderheiten einzelner Geräte abstrahieren, so daß das Betriebssystem dem Benutzer eine einheitliche Schnittstelle zu allen unterstützten Geräten anbieten kann.

Gerätetreiber für OS/2.

Unter OS/2 ist der Zugriff auf Geräte noch ein wenig komplizierter, da innerhalb des Betriebssystems mehrere Schnittstellen zu den Geräten existieren. Der von OS/2 bevorzugte Zugriff auf Geräte ist in Abbildung 4.5 vereinfacht dargestellt. Zwischen der eigentlichen Anwendung und dem Gerät gibt es mehrere Schnittstellen, die den Zugriff auf das Gerät koordinieren (gestrichelt umrandet). Die 'Subsystem Funktionsaufrufe' erlauben der Anwendung den asynchronen Zugriff auf das Gerät. Diese Funktionsaufrufe werden wiederum in 'I/O-Kontroll'-Kommandos umgesetzt, die im wesentlichen einer Funktion des Gerätetreibers entsprechen. Die 'Paket-Schnittstelle' bereitet die übergebenen Daten noch einmal für den Gerätetreiber auf, bevor dieser letztendlich auf das jeweilige Gerät zugreifen kann. Der Programmierer kann bei dieser Art der Realisierung über ein "generisches Ein- und Ausgabekommando" (vgl. [Mue94, S.

[19] Beispiele für solche Ergebnisse wären: 'Mittlere Dauer einer Verbindung', 'Anzahl der Verbindungen zu einem Knoten', 'Summe der Verbindungsabbrüche während der Messung' usw.

[20] Das Betriebssystem unterstützt die Programmierung von Anwendungen durch diverse Funktionen, u.a. auch durch Funktionen zur Ansteuerung der (Peripherie-) Geräte. Der 'natürliche Platz' für die Funktionen zur Ansteuerung der 'Transputerboards' wäre daher im Betriebssystem des 'Hostrechners'.

357ff]) auf das Gerät zugreifen. Eine ausführlichere Darstellung der Einbettung von Gerätetreibern in das Betriebssystem OS/2 wird in [Int89] gegeben.

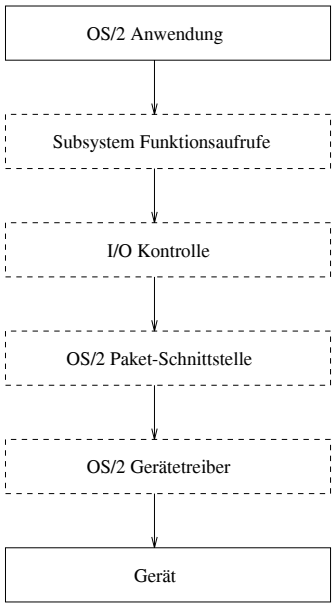


Abbildung 4.5: Zugriff einer OS/2 Anwendung auf Geräte

Alternativ kann man unter OS/2 auf Geräte durch 'privilegierte' Befehle[21] zugreifen. Der Anwendungsprozeß greift dabei ohne Betriebssystemunterstützung direkt auf die Ein- und Ausgabeadressen der Geräte zu. Der Hauptnachteil dieser Methode ist, daß verschiedene Anwendungen jeweils ihre eigene Geräteunterstützung realisieren.

Die Realisierung eines Gerätetreibers für die 'Transputerboards' wurde verworfen, da hiermit ein enormer Programmieraufwand verbunden ist. Außerdem müßten die Zugriffe auf das Gerät 'Transputerboard' dann den Aufrufkonventionen des Subsystems entsprechen. Um ein eigenes Format zu benutzen, das besonders für die Kommunikation mit Transputern geeignet ist, wäre eine weitere Schnittstelle für die entsprechende Umsetzung zu programmieren. Ein weiterer Nachteil dieser Lösung ist der relativ hohe 'Overhead', der mit dem Zugriff auf Geräte über Gerätetreiber einhergeht. Neben den oben dargestellten sukzessiven Funktionsaufrufen für jede Schnittstelle muß man noch den Aufwand für mindestens zwei Prozeßwechsel hinzurechnen[22].

[21] Das Anwendungsprogramm hat durch 'privilegierte' Befehle kurzzeitig die gleichen (universellen) Zugriffsrechte wie der Betriebssystem-Kern. Daher lösen diese direkten Zugriffe keine Schutzverletzungen aus.

[22] Mindestens ein Prozeßwechsel beim Übergang von der Anwendung zum 'Subsystem' (als Teil des Betriebssystems) und ein weiterer Prozeßwechsel beim Übergang vom Betriebssystem zurück zur Anwendung. Dabei ist zu beachten, daß die Prozeßwechsel unter OS/2 um ein vielfaches aufwendiger sind, als auf einem Transputer.

Die Ansteuerung der 'Transputerboards' unter OS/2 soll durch 'privilegierte' Befehle erfolgen. Zur Programmlaufzeit müssen mindestens die folgenden Zugriffe auf das 'Transputerboard' möglich sein:

- Prüfen der Sendebereitschaft des ersten Transputers,
- Prüfen der Empfangsbereitschaft des ersten Transputers,
- Senden von Nachrichten an die Transputer,
- Empfangen von Nachrichten von den Transputern.

Auch in diesem Fall soll das Empfangen von Nachrichten vom Transputer nebenläufig zu anderen Berechnungen auf dem Host erfolgen. Das Senden von Nachrichten erfolgt im wesentlichen als Reaktion auf Benutzereingaben und muß daher nicht nebenläufig realisiert werden.

4.6 Sonstige unterstützende Funktionen

Neben den oben dargestellten Funktionen werden eine Reihe weiterer Funktionen zur Unterstützung benötigt. Diese Funktionen braucht man jedoch nicht explizit zu entwerfen, da es für sie entweder 'Standardentwürfe' gibt oder eine Realisierung 'ad hoc' — aufgrund der geringen Komplexität — möglich ist. Die wichtigsten Funktionen dieser Kategorie werden kurz vorgestellt:

4.6.1 Funktionen zur Ringpuffer-Verwaltung

Um kurzzeitige Spitzenlasten (durch viele Nachrichten) ausgleichen zu können, benötigen die Vermittlungsfunktionen (s.o.) Ringpuffer. Eintreffende Nachrichten können in Ringpuffern bis zur Verarbeitung kurzzeitig zwischengespeichert werden. Das explizite Reservieren und Freigeben von dynamischen Speicher für eintreffende Nachrichten ist unter hoher Last zu langsam[23].

4.6.2 Funktionen zur Symboltabellen-Verwaltung

Vordefinierte und benutzerdefinierte Symbole sollen in einer Symboltabelle verwaltet werden. Die dafür benötigten Funktionen sind typischerweise

- Eintragen und Löschen von Symbolen,
- Auffinden von Symbolen und deren Attributen und
- Ändern der Attribute eines Symbols.

[23] Die Ausführungsgeschwindigkeit der Standardfunktionen für die dynamische Speicherverwaltung in "C" (malloc() und free()), vgl. [KR92, S. 177ff]) ist von mehreren Faktoren abhängig. Die Leistungsfähigkeit des Prozessors, die Implementation der Funktionen und der Grad der Fragmentierung der 'Halde', von dem der Speicher genommen wird, sind Beispiele für diese Faktoren. Durch eigene Messungen konnte gezeigt werden, daß diese Funktionen in der benutzten Entwicklungsumgebung zu langsam sind.

4.6.3 Funktionen zum Umwandeln von Daten

Es müssen diverse Umwandlungsfunktionen realisiert werden. Die Umwandlung von 'IP'-Adressen in Zeichenketten und 'vice versa' ist ein Beispiel. Ein anderes Beispiel ist die Umwandlung von (Adress-)Wörtern aus der 'Network Byte Order' (NBO)[24] in das Fromat, das von Transputern und 'Hostrechner'-Prozessor verlangt wird.

Da Transputer und der Prozessor des 'Hostrechners' die Daten in der gleichen Reihenfolge im Speicher ablegen[25], können Daten zwischen Transputer und 'Hostrechner' ohne Umwandlung ausgetauscht werden. Leider entspricht diese Reihenfolge nicht der NBO der TCP/IP-Protokollfamilie[26]. Es werden Umwandlungsfunktionen für 16-Bit-Wörter und 32-Bit-Wörter benötigt.

Derartige Umwandlungsfunktionen sind sehr leicht realisierbar (allgorithmisch einfach). Da diese Funktionen jedoch häufig aufgerufen werden, ist eine schnelle Implementation erforderlich. Typische Realisierungen verwenden 'Makros', durch die der Programmcode immer an die entsprechende Stelle im Programm kopiert wird. Diese Vorgehensweise ist schneller als die Verwendung von Funktionsaufrufe (vgl. beispielsweise [CS91, S.581]). Die effiziente Übersetzung muß vom Compiler vorgenommen werden. Alternativ kann man diese zeitkritischen Umwandlungsfunktionen in der Maschinensprache des Prozessors programmieren. In diesem Fall liegt die 'Verantwortung' für die Effizienz der Realisierung beim Programmierer.

4.7 Klassifikation der Funktionen

In diesem Kapitel wurden verschiedene Funktionen entworfen. Einige dieser Funktionen wurden als *essenziell* bezeichnet, da sie unmittelbar der Realisierung der Benutzeranforderungen dienen. Die *unterstützenden* Funktionen bilden die Plattform, auf der die essentiellen Funktionen aufbauen. Ihre Funktionalität soll von den essentiellen Funktionen getrennt werden, um die

- Modularität zu erhöhen,
- Realisierung und Erweiterungen zu vereinfachen und
- (Programm-)Redundanz zu vermeiden.

Eine weitere Möglichkeit, die Funktionen einzuteilen, ergibt sich, wenn man von den Restriktionen durch die Hardware und dem gewählten Paradigma der 'Paketparallelität' ausgeht. Bei dieser Sicht ergibt sich, daß viele Funktionen entworfen wurden, um die Hardware anzusteuern oder die Analyse nach dem Paradigma der Paketparallelität zu ermöglichen. Funktionen, die keiner dieser beiden Kategorien angehören,

[24] Die 'Network Byte Order' legt fest, in welcher Reihenfolge die Byte eines Wortes übertragen werden. Eine Festlegung ist notwendig, da nicht alle Prozessoren die Byte eines Wortes in der gleichen Reihenfolge ablegen.

[25] Bei beiden Prozessoren handelt es sich um sogenannte 'Little Endian'-Maschinen, die das niederwertigste Byte eines Wortes an der Speicherstelle mit der kleinsten Adresse ablegen.

[26] Das höchstwertigste Byte eines Wortes wird zuerst übertragen. Das entspricht dem 'Big Endian'-Format.

wurden entworfen, um direkt den Anforderungen zu genügen. Bei der folgenden Klassifikation (vgl. Tabelle 4.1) ist zu beachten, daß die Zugehörigkeit zu einer Kategorie den *wesentlichen Aspekt* einer Funktion kennzeichnet. Damit ist nicht ausgeschlossen, daß eine Funktion auch unter einer anderen Kategorie eingeordnet werden könnte.

Einige Funktionen des Meßwerkzeuges		
	essentielle Funktionen	unterstützende Funktionen
Benutzeran-forderungen	Deklaration von Filtern Filterfunktionen Statistikfunktionen Fkt. zum Verschriftl. Exportfilter	Umwandlungsfunktionen
'Paketparallelität'	globale Statistikfkt.	Verteilungsfunktionen Verbindungsmana-gementfunktionen
Hardware	Gerätetreiber Ethernet (Rohdatenerfassung)	Vermittlungsfunktionen Gerätetreiber TB Umwandlungsfunktionen

Tabelle 4.1: Klassifikation der Funktionen

Kapitel 5

Realisierung eines parallelen Meßwerkzeuges für Ethernet-basierte Netze

In diesem Kapitel wird die Implementierung des Meßwerkzeuges beschrieben. Es wird dargestellt, wie die Anforderungen aus Abschnitt 2.7 unter Beachtung der Restriktionen aus Abschnitt 2.7.4 und 4.1 auf Grundlage des Konzeptes der Paketparallelität (vgl. Abschnitt 3.3.3.5) realisiert werden.

In Abschnitt 5.1 wird Abbildung 4.1 wieder aufgegriffen. Durch den Entwurf der Funktionen in Kapitel 4 ist es möglich, die Funktionen Prozessen zuzuordnen, so daß sich eine gegenüber Abbildung 4.1 verfeinerte Darstellung ergibt. Die einzelnen Prozesse werden benannt und auf Grundlage eines allgemeinen 'Prozeß-Templates' verfeinert (vgl. Abschnitt 5.2).

In den Abschnitten 5.3 und 5.4 werden schließlich die Realisierungen der einzelnen Funktionen diskutiert. Die Darstellungsreihenfolge orientiert sich am Entwurf aus Kapitel 4.

Das Vorgehen bei Erweiterungen wird in Abschnitt 5.6 diskutiert. Als Beispiel für die Erweiterung des Prototypen wird die Vorgehensweise zur Implementierung einer neuen Prozeßklasse dargestellt. Außerdem werden in diesem Abschnitt sinnvolle Erweiterungen diskutiert.

In Abschnitt 5.7 wird der Prototyp mit anderen Werkzeugen der Kommunikationsmeßtechnik verglichen.

Das dynamische Verhalten der Konfiguration wird anschließend in Kapitel 6 beschrieben. Anhand von Beispielsmessungen wird gezeigt, wie das Meßwerkzeug gestartet, initialisiert und für verschiedene Meßaufgaben eingesetzt wird.

5.1 Zusammenstellen der Funktionen zu Prozessen

In diesem Abschnitt wird dargestellt, wie die essentiellen Funktionen und Unterstützungsfunktionen zu Prozessen zusammengestellt werden. Unter einem *Prozeß* wird hier

eine Menge von Funktionen, die sich einen gemeinsamen Datenbereich teilen ('Umgebung') und mindestens ein Kontrollfluß — auch Ablauffaden ('Thread') genannt[1] —, der die Reihenfolge der Funktionsaufrufe angibt, verstanden[2]. Ein Prozeß kann auch über mehrere (nebenläufige) 'Threads' verfügen. Der Unterschied zwischen einem Prozeß und einem 'Thread' ist, daß die 'Threads' sich die Umgebung ihres Prozesses teilen, jeder Prozeß hingegen über eine eigene Umgebung verfügt, die typischerweise vom Betriebssystem vor den Umgebungen anderer Prozesse geschützt wird.

Die grundlegende Entwurfsentscheidung, nur einen Prozeß auf jeden Transputer zu plazieren, wird im folgenden Abschnitt erläutert.

5.1.1 Prozesse und 'Threads' auf Transputern

Ein Transputer ist in der Lage, Prozesse ohne ein Betriebssystem zu verwalten (vgl. Abschnitt 3.2.1.2). Auch die geringen Zeiten für einen Prozeßwechsel (ca. $1\mu s$) lassen Diskussionen[3], ob 'Threads' oder Prozesse auf Transputern ausgeführt werden sollten, irrelevant erscheinen.

Da aufgrund des fehlenden Betriebssystems die Betriebsmittelverwaltung auf den Transputern[4] jedoch von den (Anwendungs-)Prozessen selbst übernommen werden muß, bietet sich die Verwendung *eines Prozesses mit mehreren 'Threads' pro Transputer* an. So können sich verschiedene 'Threads' beim Zugriff auf die Betriebsmittel aufwandsarm über globale Variablen synchronisieren. Außerdem entfällt die Realisierung von Management-Prozessen für die oben genannten Betriebsmittel.

Im folgenden werden demnach Funktionen zu 'Threads' zusammengestellt. Alle 'Threads' eines Transputers gehören zum selben Prozeß. Die gesamten Betriebsmittel eines Transputers werden von diesem Prozeß verwaltet.

5.1.2 Darstellen der Prozesse mit Datenflußdiagrammen

In den folgenden Abschnitten wird dargestellt, wie Funktionen zu 'Threads' zusammengestellt werden. Die Nebenläufigkeit innerhalb der einzelnen Prozesse und der Prozesse untereinander wird dadurch hinreichend dargestellt.

---

[1] Allgemeine Definitionen von 'Thread' und 'Prozeß' werden in [Tan95, S. 205ff] und [CDK94, S. 165ff] gegeben. Eine spezifische Darstellung am Beispiel des Betriebssystems 'Mach' wird z.B. in [ZK93] gegeben.

[2] Die in Abschnitt 3.1.1 gemachte Definition der Prozesse, als Realisierung nebenläufiger Teilalgorithmen eines Programms, ist dadurch nicht aufgehoben. Vielmehr wird die Definition hier um einen operationalen Aspekt erweitert. Jeder Teilalgorithmus wird auf einer Funktionseinheit (einem Prozessor) ausgeführt. Für den Programmcode des Algorithmus und die Datenstrukturen auf denen er arbeitet, muß Speicherplatz (in der 'Umgebung') zur Verfügung stehen. Die Ausführung des Programmcodes durch den 'Thread' ist die Abarbeitung des Teilalgorithmus. Somit repräsentieren 'Thread' und 'Umgebung' den Teilalgorithmus als Prozeß auf einem Prozessor.

[3] Das Hauptargument, wenn möglich 'Threads' zu verwenden, beruht auf der Beobachtung, daß ein 'Thread'-zu-'Tread'-Wechsel innerhalb einer Umgebung weniger 'Overhead' produziert, als ein Prozeßwechsel.

[4] Die wichtigsten Betriebsmittel des Transputers sind neben der Prozessorzeit die Transputerlinks und der physikalische Speicher.

Entscheidend an der Paketparallelität ist jedoch der Fluß der Dateneinheiten durch die Paketträger. Daher wird für jeden Prozeß eine graphische Darstellung des Datenflusses durch die Funktionen bzw. 'Threads' gegeben. Besonders geeignet für eine Darstellung des Datenflusses durch Funktionseinheiten sind die sogenannten 'Datenflußdiagramme' (vgl. [You89]). Sie ermöglichen auf Grundlage weniger graphischer Symbole die Darstellung nebenläufiger Funktionseinheiten, Speicher und der auszutauschenden Daten.

Die graphischen Symbole der Datenflußdiagramme

In Abbildung 5.1 sind alle graphischen Symbole aufgeführt, die in einem Datenflußdiagramm vorkommen können. Wesentlich hierbei sind die folgenden Symbole:

- Funktionen, Prozesse und Funktionseinheiten teilen sich dasselbe Symbol. Die entsprechenden Symbole stellen *aktive Komponenten* dar, die die Daten verarbeiten und weiterleiten. Neben einem Namen werden diese Symbole zusätzlich mit einer Nummer markiert, die die gegenwärtige Hierarchiestufe ausdrückt. Hat eine Funktion auf der obersten Hierarchiestufe die Nummer x, so sind beispielsweise $x.y$ und $x.z$ $(x, y, z \in N)$ Verfeinerungen dieser Funktion auf der darunterliegenden Hierarchiestufe.
- Datenfluß. Der durch Pfeile dargestellte Datenfluß beschreibt Daten, die von einer aktiven Komponente zu einer anderen übertragen werden, bzw. von oder zu einem Speicher übertragen werden.
- Speicher. Speicher lagern Daten, bis sie von einer aktiven Komponente verarbeitet werden.
- Terminatoren. Terminatoren sind die Schnittstellen zum modellierten System. Terminatoren und Wechselwirkungen der Terminatoren untereinander werden nicht modelliert. Über Terminatoren werden Daten oder Signale an das System abgegeben bzw. vom System an die Umgebung abgegeben.
- Kontrollkomponenten und Kontrollfluß. Kontrollkomponenten senden bzw. empfangen Signale. Durch Kontrollkomponenten wird im wesentlichen das zeitliche Verhalten des Modells beschrieben.

Für eine vertiefende Darstellung vgl. [MP86] oder [You89].

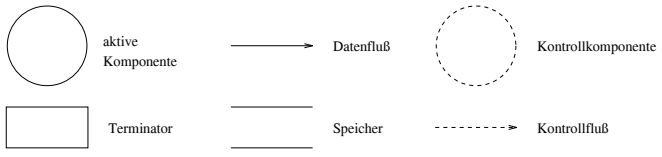


Abbildung 5.1: Symbole der Datenflußdiagramme

5.1.3 Die Prozesse des Transputernetzes als Datenflußdiagramm

Die Abbildung 4.1 wird hier als Datenflußdiagramm dargestellt. Es handelt sich hierbei um eine Verfeinerung der Abbildung 4.1. Da genau ein Prozeß auf jeden Transputer plaziert wird, werden die Prozessoren in dieser Abbildung nicht mehr dargestellt. Der 'Hostrechner'-Prozeß soll nicht modelliert werden. Er wird deshalb als Terminator dargestellt. Ein weiterer Terminator ist die Ethernet-Schnittstelle. Das so abgegrenzte System ist in Abbildung 5.2 dargestellt. Man beachte die Numerierung der Prozesse, die in den folgenden Abschnitten verfeinert werden. Der Kontrollfluß wird in dieser und den folgenden Darstellungen vernachlässigt.

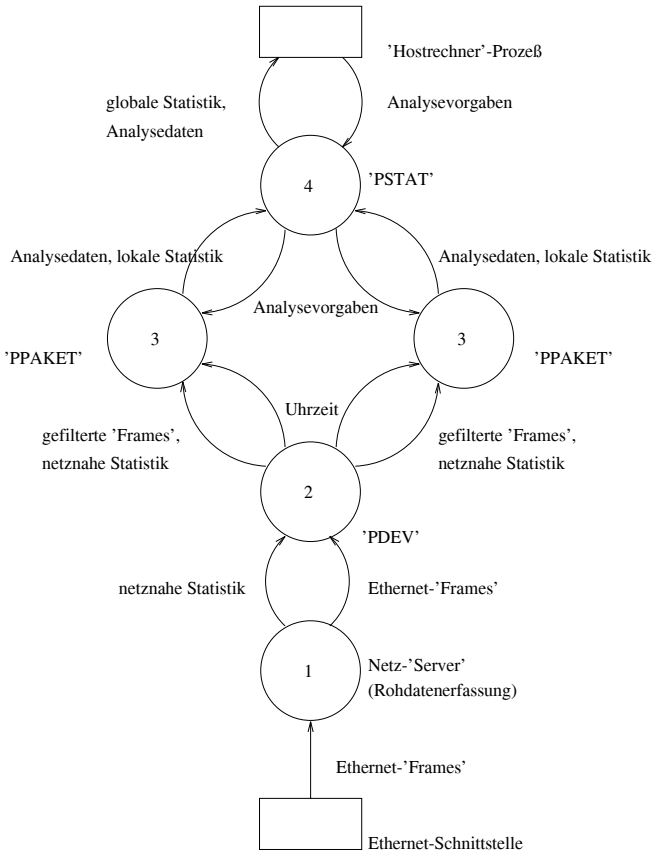


Abbildung 5.2: Datenflußdiagramm der Transputerprozesse

Zum zweiten Prozeß ('PDEV') ist anzumerken, daß hier die Filterfunktionen ausgeführt werden. Die Entscheidung, das Filtern von den Paketträgern zu trennen (vgl. Abschnitt 4.4.5.1), läßt sich dadurch begründen, daß der Prozessor 'PDEV' allein durch das Zwischenspeichern von Rohdaten kaum ausgelastet wird. Außerdem wird durch das frühzeitige Filtern die Last der Verteilungsfunktionen, die notwendigerweise von 'PDEV' ausgeführt werden, verringert. Sie brauchen nur noch die Dateneinheiten

auf die Paketträger zu verteilen, die die Filterfunktionen 'übriglassen'.

5.2 Verfeinerung der Prozesse

In diesem Abschnitt werden die Prozesse aus Abbildung 5.2 verfeinert. Die Grundlage für diese Verfeinerung bildet ein allgemeines 'Prozeß-Template', das die Grobstruktur aller Prozesse darstellt, die auf den Transputern ausgeführt werden (vgl. Abschnitt 5.2.1).

5.2.1 Ein allgemeines 'Prozeß-Template'

Jeder Prozeß muß in der Lage sein, Nachrichten zu senden und zu empfangen. Die dafür notwendigen Vermittlungsfunktionen wurden in Abschnitt 4.5.1 entworfen. Es wurden außerdem die Vorteile diskutiert, Sende- und Empfangsfunktionen voneinander und von den verbleibenden Funktionen zu trennen. Es liegt nahe, diese drei Funktionsklassen als eigenständige 'Threads' zu realisieren. Um das (ineffiziente) Kopieren von Nachrichten zu vermeiden, sollen alle 'Threads' Zugriff auf einen gemeinsamen Ringpuffer haben. Die allgemeine Struktur eines Transputer-Prozesses ist in Abbildung 5.3 als Datenflußdiagramm dargestellt.

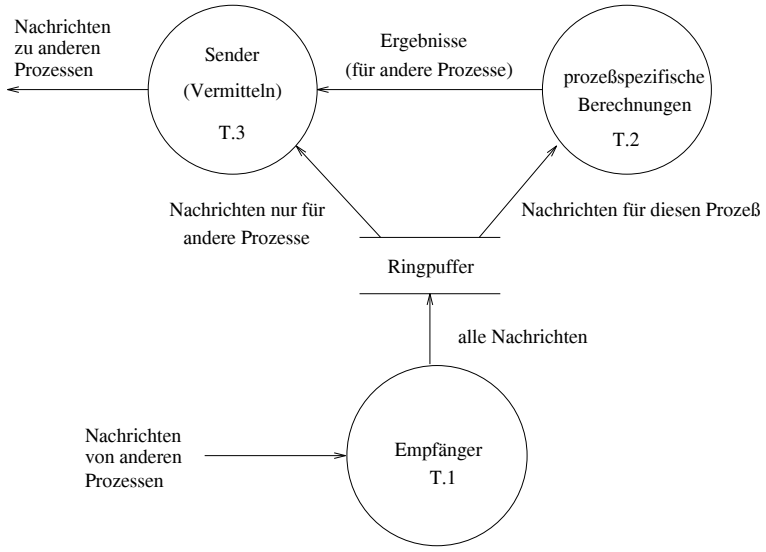


Abbildung 5.3: Datenflußdiagramm des 'Prozeß-Templates'

Die Kooperation dieser 'Threads' wird weiter unten am Beispiel des Prozesses 'PDEV' exemplarisch dargestellt. Aus der Darstellung ist ersichtlich, daß alle Prozesse, die auf einem Transputer plaziert werden, in ihrer Struktur ähnlich sind. Es gibt immer einen 'Thread', der den Ablauf koordiniert. Er startet die anderen 'Threads' des Prozesses. Dies sind in jedem Fall die 'Threads' für die Nachrichtenvermittlung.

Weitere 'Threads' werden beispielsweise bei 'PDEV' (vgl. Abschnitt 5.2.3) gestartet. Alle Aktionen nach der Initialisierung durch die Ablaufkoordinierung werden durch Nachrichten ausgelöst.

5.2.2 Der Prozeß zur Rohdatenerfassung

Hier sind die Funktionen des Gerätetreibers für die Ethernet-Schnittstelle zusammengefaßt worden (vgl. Abschnitt 4.4.1). Eine Besonderheit dieses Prozesses ist, daß er bei der Konfigurierung nicht auf dem Transputer P6 (vgl. Abbildung 4.1) plaziert wird. Dieser Prozeß muß 'nachträglich' durch eine Funktion, die auf P5 abläuft, auf den Transputer P6 geladen werden. Außerdem kann aufgrund des geringen Speichers, der diesem Prozeß auf P6 (vgl. Abbildung 4.1) zur Verfügung steht, nur eine — gegenüber dem 'Prozeß-Template' stark vereinfachte — Struktur realisiert werden. Hauptaufgabe dieses Prozesses ist es, empfangene Dateneinheiten an den Prozeß 'PDEV' (s.u.) weiterzuleiten und netznahe Statistiken zu generieren.

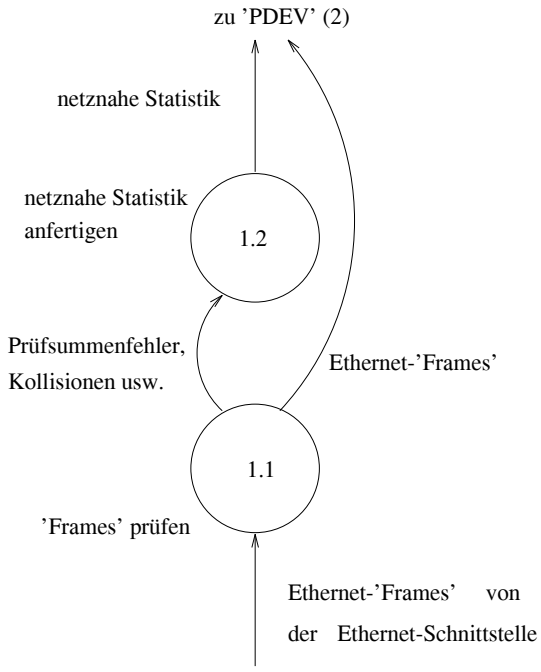


Abbildung 5.4: Datenflußdiagramm des Netz-'Servers'

5.2.3 Der Prozeß 'PDEV'

Der Prozeß 'PDEV' soll auf den Transputer 'P5' (vgl. Abbildung 4.1) plaziert werden. Die Hauptaufgabe dieses Prozesses ist das Empfangen, Filtern und Verteilen der Dateneinheiten, die von der Rohdatenerfassung an ihn gesendet werden.

5.2.3.1 'Thread' zum Filtern und Verteilen von Dateneinheiten

Offensichtlich werden dazu die Filterfunktion und die Verteilungsfunktion (vgl. Abschnitt 4.5.1) benötigt. Diese beiden Funktionen werden innerhalb eines 'Threads' ausgeführt, da sie immer nacheinander ausgerufen werden müssen. Eine Dateneinheit wird beim Filtern entweder vernichtet ('herausgefiltert') oder als zu analysierende Dateneinheit mit einem Zeitstempel und einer Ereignisnummer an die Verteilungsfunktion weitergeleitet.

5.2.3.2 'Thread' zum Empfangen der Dateneinheiten

Der Empfang von Dateneinheiten wird durch einen eigenen 'Thread' realisiert, der als einer von zwei 'Threads' auf hoher Priorität abläuft. Durch die hohe Priorität wird dieser 'Thread' sofort vom Prozessor ausgeführt, wenn eine Dateneinheit von der Rohdatenerfassung gesendet wird. Dieses Vorgehen ist notwendig, um nach Möglichkeit ein Überlaufen[5] der viel zu klein dimensionierten Empfangswarteschlange auf P6 zu vermeiden. Die empfangenen Dateneinheiten werden in einer als Ringpuffer organisierten Warteschlange gespeichert.

5.2.3.3 'Threads' für den Nachrichtenaustausch

Die Vermittlungsfunktionen (vgl. Abschnitt 4.5.1) werden auf vier 'Threads' aufgeteilt. Ein 'Thread' ist für den Empfang von Nachrichten zuständig, die ebenfalls in einem Ringpuffer abgelegt werden. Ein weiterer 'Thread' übernimmt das Vermitteln der Nachrichten. Nachrichten, die Daten für eine interne[6] Funktion übermitteln, werden von dem dritten 'Thread', dem Kontroll-'Thread', verarbeitet. Unabhängig davon kann jeder 'Thread' Nachrichten versenden. Die dafür realisierte Funktion mußte als Monitor[7] implementiert werden, da sie über einen internen Kanal (in diesem Fall das zu schützende Betriebsmittel) mit dem 'Thread' zum Senden von Nachrichten kommuniziert.

5.2.3.4 'Thread' für die Ablaufkoordinierung

Der Kontroll-'Thread' reagiert im wesentlichen auf interne Nachrichten (vgl. Abbildung 5.3, 'prozeßspezifische Berechnungen'). Er koordiniert die Kooperation der anderen 'Threads' des Prozesses. Dazu startet er nicht nur die anderen 'Threads' im Zuge der Initialisierung, sondern initialisiert gemeinsame Variablen und teilt interne

[5] Ein Überlaufen der Empfangswarteschlange auf P6 bedeutet einen Datenverlust. Dateneinheiten, die aufgrund einer vollen Warteschlange nur teilweise oder gar nicht empfangen werden konnten, stehen für die Messung nicht zur Verfügung. Im schlimmsten Fall sind dadurch die Meßergebnisse falsch.

[6] Eine interne Funktion gehört zum aktuellen Prozeß. Eine externe Funktion gehört zu einem anderen Prozeß. Aufgrund der oben diskutierten Plazierung maximal eines Prozesses auf jeden Transputer, ist eine externe Funktion auch immer auf einem anderen Prozessor.

[7] Ausnahmsweise wird diesmal mit Monitor eine Funktion bezeichnet, die durch eine Semaphore geschützt wird, so daß nur ein 'Thread' zur Zeit den 'Funktionscode' ausführen kann.

Kommunikationskanäle den anderen 'Threads' zu. Ein Beispiel ist die Reaktion auf die 'Status-Anforderung'. Diese Nachricht wird typischerweise vom 'Hostrechner' mit einer 'Broadcast'-Nachricht an alle Prozesse gesendet. Der Kontroll-'Thread' stellt nun eine prozeßspezifische Nachricht zusammen, die Auskunft über bestimmte Zustandsvariablen des Prozesses gibt. Diese Nachricht wird dann an den 'Hostrechner' zurückgesendet.

Die Kooperation zwischen dem Kontroll-'Thread' und den 'Threads', die die Nachrichtenvermittlung realisieren, ist bei allen Prozessen gleich. Unterschiede bestehen lediglich in der Anzahl und der Art von Nachrichten, auf die reagiert wird bzw. an welche Prozesse sie weitervermittelt werden. Bei der Beschreibung der folgenden Prozesse wird daher nicht mehr auf diese 'Threads' eingegangen.

5.2.3.5 'Thread' für die Aktualisierung des Zeitstempels

Dieser 'Thread' ist ebenfalls hochpriorisiert. Er hat zwei Aufgaben:

- Aktualisieren des Zeitstempels mit einer Auflösung von einer Millisekunde. Der Zeitstempel wird in einer globalen Variable gespeichert und von der Filterfunktion verwendet, um Dateneinheiten zu markieren, die nicht vernichtet werden.
- Periodisches Auslösen einer Nachricht 'TIMER', nach Ablauf einer vom Benutzer wählbaren Zeit. Beim Prototypen wird diese Nachricht jede Sekunde ausgelöst und an die Paketträger weitervermittelt. Die Paketträger nutzen diese Nachricht, um periodisch ihre lokalen Statistiken an die globale Statistik weiterzuleiten.

Alternativ hätte auf jedem Prozessor, auf dem ein Paketträger plaziert wird, ein eigener Zeitgeber realisiert werden können. Das gewählte Vorgehen ist jedoch eleganter und aufwandsärmer, da die Paketträger durch die 'TIMER'-Nachricht synchronisiert werden. Ihre Reaktion auf diese Nachricht kann quasi gleichzeitig erfolgen. Bei autonomen Zeitgebern auf jedem Paketträger müßte man Mechanismen zum Abgleichen der Uhren realisieren. Das Datenflußdiagramm des Prozesses ist in Abbildung 5.5 gegeben.

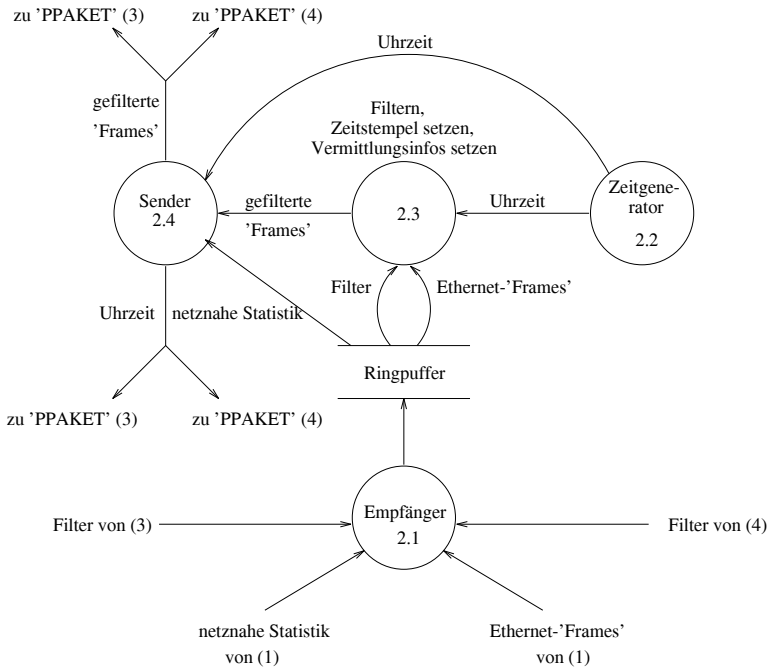


Abbildung 5.5: Datenflußdiagramm von 'PDEV'

5.2.4 Der Prozeß 'PPAKET'

Bei 'PPAKET' handelt es sich um den bereits mehrfach erwähnten Paketträger. Jeweils eine Instanz dieses Prozesses wird auf P3 und P4 plaziert. Es handelt sich um exakt gleichartige Prozesse, die sich lediglich in der Auswahl ihres Eintrags aus der Vermittlungstabelle (vgl. Abschnitt 4.5.1) unterscheiden. Während ein Prozeß Nachrichten gegebenenfalls von P1 und P2 an P5 und P6 weiterleitet, vermittelt der andere die Nachrichten in die entgegengesetzte Richtung. Neben den obligatorischen 'Threads' zur Nachrichtenvermittlung gibt es nur noch einen weiteren 'Thread'. Er übernimmt sowohl die Kontrollfunktionalität als auch die Analyse der Dateneinheiten.

'Thread' für Analyse und Ablaufkoordinierung

Die Ablaufkoordinierung bearbeitet alle Nachrichten, die von anderen Prozessen an den Paketträger gesendet werden. Da die Dateneinheiten als Nachrichten 'verpackt' vom Prozeß 'PDEV' beim Paketträger eintreffen, veranlaßt die Ablaufkoordinierung auch die Analyse der Dateneinheiten[8].

[8] Ein eigenständiger 'Analyse-Thread' hätte nur sehr geringe Vorteile. Ein Beispiel wäre die schnellere Reaktion auf Nachrichten, die keine Dateneinheiten transportieren ('Kontrollnachrichten'). Während der 'Analyse-Thread' noch beschäftigt ist, könnte die Ablaufkoordinierung den Ringpuffer vorausschauend nach Kontrollnachrichten durchsuchen.

Neben dem Auswerten der eintreffenden Dateneinheiten veranlaßt die Ablaufkoordinierung das Versenden der lokalen Statistiken an die globale Statistik (s.u. 'PSTAT'), als Reaktion auf die von 'PDEV' versendete 'TIMER'-Nachricht (s.o. 'PDEV'). Eine weitere wichtige Nachricht wird vom 'Hostrechner' an die Paketträger gesendet. Immer wenn der Benutzer eine Auswahl der zu analysierenden Protokolle trifft, muß die Wahl den Paketträgern mitgeteilt werden[9].

Die meisten Nachrichten beinhalten jedoch Dateneinheiten, die vom Paketträger analysiert werden. Die Analyseergebnisse jeder Dateneinheit werden dann an den 'Hostrechner' gesendet (vgl. Abbildung 5.6).

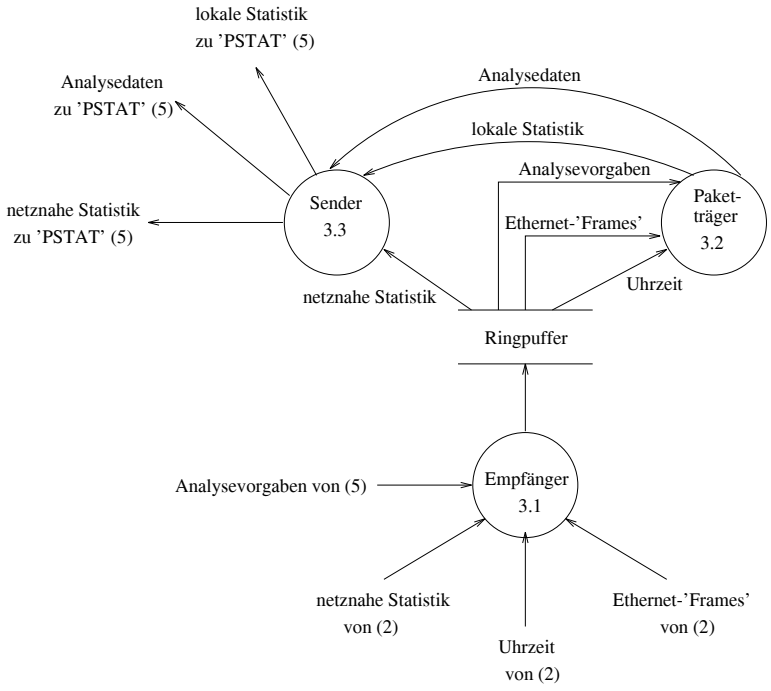


Abbildung 5.6: Datenflußdiagramm von 'PPAKET'

5.2.5 Der Prozeß 'PSTAT'

Der Prozeß 'PSTAT' berechnet die globalen Statistiken auf Basis der Meßdaten, die ihm periodisch von den Paketträgern gesendet werden. Er ist ähnlich simpel organisiert wie die Paketträger. Neben den obligatorischen 'Threads' für die Nachrichtenverwaltung gibt es auch hier nur einen weiteren 'Thread'.

[9] Auf die Informationen dieser 'Auswahl'-Nachricht wird bei der Analyse ständig zugegriffen. Sie ermöglichen nicht nur die Entscheidung, ob Dateneinheiten eines bestimmten Protokolls überhaupt analysiert werden sollen (vgl. Abbildung 5.9), sondern auch die Selektion der Protokollparameter, die der Benutzer für die Analyse ausgewählt hat.

5.2.5.1 'Thread' für Statistik und Ablaufkoordinierung

Die Ablaufkoordinierung reagiert im wesentlichen auf die Nachrichten mit den lokalen Statistiken, die periodisch von den Paketträgern eintreffen. Beim Prototypen werden lediglich diese lokalen Statistiken zu einer globalen Statistik zusammengefaßt und an den 'Hostrechner' gesendet (vgl. Abschnitt 4.4.3).

An dieser Stelle ist viel Raum für zahlreiche weitere Statistiken, beispielsweise Berechnungen zur Längenverteilung der Dateneinheiten, periodisches Erstellen von Verkehrsmatrixen etc.

Bei ersten Messungen mit dem Prototypen hat sich herausgestellt, daß die Schnittstelle vom 'Hostrechner' zum TB (vgl. Abbildung 3.7) der primäre Engpaß bei den Messungen ist. Aus diesem Grund wurden außerdem Überwachungsfunktionen realisiert, die dem Benutzer gegebenenfalls mitteilen, daß Analysedaten oder Dateneinheiten verlorengehen.

Eine dieser Überwachungsfunktionen wird von der Ablaufkoordinierung von 'PSTAT' aufgerufen. Bevor die Ablaufkoordinierung eine neue Nachricht bearbeitet, überprüft sie dazu eine Flagge, die von dem Empfangs-'Thread' gesetzt wird, wenn er aufgrund der Füllung des Ringpuffers keine weiteren Nachrichten ablegen kann. In diesem Fall wird der Benutzer benachrichtigt. Auch 'PSTAT' ist als Datenflußdiagramm in Abbildung 5.7 dargestellt.

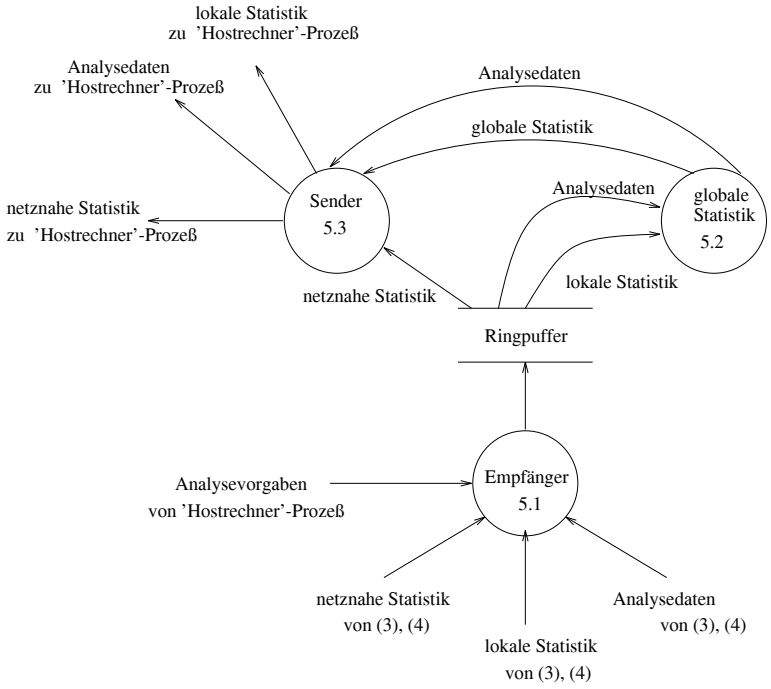


Abbildung 5.7: Datenflußdiagramm von 'PSTAT'

5.2.6 Der 'Hostrechner'-Prozeß

In Abschnitt 5.1.1 wurde dargestellt, daß es pro Transputer nur einen Prozeß gibt. Dies gilt nicht für den 'Hostrechner'. Hier übernimmt das Betriebssystem OS/2 die Betriebsmittelverwaltung und die Zuteilung der Rechenzeit. Es können (quasi-)gleichzeitig mehrere Prozesse laufen. Die Unterscheidung zwischen 'Thread' und Prozeß bleibt jedoch bestehen, da auch OS/2 mehrere Ablauffäden in einem Prozeß zuläßt. Der 'Hostrechner'-Prozeß (des Meßwerkzeuges) ist demnach einer von vielen Prozessen auf dem 'Hostrechner'.

Der 'Hostrechner'-Prozeß ist in zwei 'Threads' aufgeteilt. Ein 'Thread' ist ausschließlich für den Empfang von Nachrichten vom Transputernetz zuständig. Er führt die in Abschnitt 5.4.4 diskutierte Unterstützungsfunktion zum Empfangen von Nachrichten aus.

Der andere 'Thread' ist wiederum die Ablaufkoordinierung. Er hat folgende Aufgaben:

- Initialisieren des Werkzeuges. Einerseits müssen Initialisierungen für die in Abschnitt 5.5 beschriebenen Funktionen durchgeführt werden. Insbesondere muß der Ringpuffer für Nachrichten eingerichtet werden und die Symbolverwaltung für den Benutzer.

  Außerdem lädt der 'Hostrechner'-Prozeß die Prozesse auf die Transputer. Dazu verwendet er die in Abschnitt 5.3.4.6 beschriebene Funktion.

- Reagieren auf Benutzereingaben. Dazu werden die Unterstützungsfunktionen aus Abschnitt 5.3.4.1 benutzt.

- Reagieren auf Nachrichten der Transputer, die vom anderen 'Thread' empfangen wurden. Im wesentlichen handelt es sich hierbei um Meßergebnisse. Diese Ergebnisse werden 'on-line' angezeigt oder durch die Unterstützungsfunktionen aus Abschnitt 5.3.4.3 weiterverarbeitet.

Zu dem 'Hostrechner'-Prozeß ist noch anzumerken, daß er sehr hohe Anforderungen an die Leistungsfähigkeit des 'Hostrechners' stellt. Das liegt vor allem an der ständigen Überwachung der Sendebereitschaft des Transputers mittels 'polling' (vgl. Abschnitt 5.4.4). Darüber hinaus ist die Verwaltung der Benutzerein- und -ausgabe über die graphische Benutzeroberfläche sehr aufwendig. Der 'Von Neumann Flaschenhals' macht sich ebenfalls bemerkbar. Die Analyseergebnisse müssen von den Transputern empfangen und im Hauptspeicher abgelegt werden. Im Zuge der Weiterverarbeitung durch die Exportfunktionen werden sie dann erneut über den Bus des 'Hostrechners' gesendet.

Aus diesem Grund ist eine gewisse 'Zurückhaltung' durch den Benutzer angebracht. So sollten möglichst keine Anwendungen zeitgleich mit dem 'Hostrechner'-Prozeß betrieben werden, insbesondere sollte der Benutzer darauf verzichten, während einer Messung eine andere Anwendung zu starten. Aufgrund der Bedienstrategie von OS/2 ist der 'Hostrechner'-Prozeß während des Ladens einer anderen Applikation nicht in der Lage, Meßdaten in ausreichender Geschwindigkeit entgegenzunehmen.

5.3 Realisierung der essentiellen Funktionen

Die Darstellung der Realisierung der essentiellen Funktionen orientiert sich an der Darstellungsreihenfolge beim Entwurf in Abschnitt 4.4.

In Abschnitt 5.2.2 wird der Gerätetreiber für die Ethernet-Schnittstelle beschrieben. Hierbei handelt es sich sozusagen um das 'Rohdatenerfassungsmodul' (vgl. Abschnitt 4.4.1).

In Abschnitt 5.3.2 wird das Analysemodul beschrieben. Auf Grundlage der 'Paketparallelität' werden eintreffende Dateneinheiten analysiert. Dabei wird u.a. auf die zur Zeit untersuchbaren Protokolle eingegangen.

5.3.1 Realisierung des Gerätetreibers für die Ethernet-Schnittstelle

Für die Realisierung dieses Werkzeuges konnte leider kein Zugang zu einem Hochgeschwindigkeitsnetz eingerichtet werden. Es mußte deshalb mit einer Schnittstelle zu einem Ethernet-Netz gearbeitet werden (vgl. Abschnitt 2.7.4).

Eine Besonderheit dieses Gerätetreibers ist die Organisation der Funktionen in 'Client' und 'Server'. Zwei 'Client'-Prozesse (Sendeprozeß und Empfangsprozeß) laufen unabhängig voneinander und unabhängig vom 'Server'-Prozeß. Der 'Server'-Prozeß stellt die Funktionen des Gerätetreibers zur Verfügung, die von den 'Clients' angefordert werden können (vgl. [SGS91]).

Der 'Server'-Prozeß muß im Verlauf der Initialisierung des Werkzeuges von einem anderen Prozeß über den Transputerlink des T225 von seinem Nachbarprozessor geladen werden (vgl. Abschnitt 5.2.2). Dies wird von einer Funktion auf 'PDEV' übernommen.

Die Aufteilung des Gerätetreibers in 'Server'- und 'Client'-Prozesse ist in Abbildung 5.8 dargestellt.

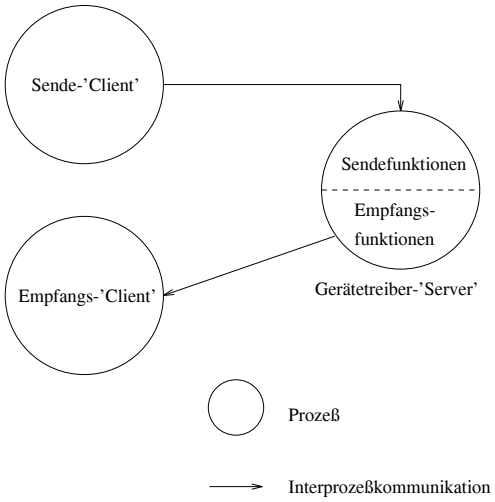


Abbildung 5.8: Gerätetreiber als 'Clients' und 'Server'

5.3.1.1 Realisierung des Empfangs-'Clients'

Ein Sende-'Client' wird zur Zeit nicht benötigt. Das Werkzeug verhält sich völlig passiv. Es beeinflußt das Netz in keiner Weise (vgl. Abschnitt 2.7.4).

Der Empfangs-'Client' muß hingegen die Initialisierung und Konfigurierung der Schnittstelle veranlassen. Dazu lädt er den 'Server' auf den T225 der Ethernet-Schnittstelle, initiiert die Selbsttest der Schnittstelle und konfiguriert die Schnittstelle. Die Funktion des Empfangs-'Client' übernimmt die Ablaufkoordinierung von 'PDEV'.

Beim Konfigurieren wird die Ethernet-Adresse der Schnittstelle eingestellt und der Empfangsmodus[10] so eingestellt, daß alle Dateneinheiten unabhängig von ihrer Empfangsadresse empfangen werden. Nur durch diese Einstellung ist es möglich, das angeschlossene Netz überhaupt beobachten und somit analysieren zu können.

Weitere Aufgaben des Empfangs-'Clients' sind:

- Ein- und Ausschalten des Empfangs bei Start und Ende einer Messung,
- Empfangen der Dateneinheiten vom 'Server' und Ablegen in eine Warteschlange,
- Anfordern und Empfangen von netznahen Statistiken und Weiterleiten an den Benutzer,
- Reagieren auf Fehlermeldungen des 'Servers'.

Die Realisierung der Funktionen für den Empfangs-'Client' bestand in einer Anpassung der Funktionsbibliothek aus [SGS91] an die verwendete Entwicklungsumgebung[11].

[10] Es handelt sich hierbei um den sogenannten 'promiscuous mode'.

[11] Es wurde ein C-Compiler der Firma 3L Ltd. eingesetzt. Die Bibliotheken zur Ansteuerung

5.3.2 Realisierung der Analysefunktionen

Im folgenden Abschnitt wird ein 'Template' einer Analysefunktion vorgestellt. Bei Erweiterungen um Analysefunktionen für neue Protokolle sollte man sich an dieses Grundgerüst halten, um mit den existierenden Analysefunktionen zusammenarbeiten zu können. In den darauffolgenden Abschnitten wird auf die speziellen Analysefunktionen für einige Protokolle der TCP/IP-Protokollfamilie eingegangen. Es handelt sich um Protokolle, die jedem Knoten bekannt sein sollten, der TCP/IP-basierte Kommunikation unterstützt. Diese Protokolle bilden den rudimentären Teil der TCP/IP-Protokollfamilie (vgl. Abschnitt 2.7.4).

5.3.2.1 'Template' einer Analysefunktion

Aufgrund des Entwurfs in Abschnitt 4.4.2 kann hier ein allgemeines 'Template' einer Analysefunktion angegeben werden. Jeder Analysefunktion müssen mindestens zwei Parameter übergeben werden. Es sind dies:

- Ein Zeiger auf die zu analysierenden Rohdaten.
- Ein Zeiger auf einen Speicherbereich, in dem die Analyseergebnisse abgelegt werden können.

Die beiden Parameter werden entweder von einer Unterstützungsfunktion bereitgestellt oder von einer anderen Analysefunktion übergeben. Da die Analyseergebnisse aller Analysefunktionen an andere Prozesse gesendet werden müssen, bietet es sich an, den Speicherbereich für alle Analyseergebnisse zusammenhängend bereitzustellen. Dadurch können nach Abschluß der Analyse einer Dateneinheit alle Ergebnisse als 'Block' versendet werden.

Eine Analysefunktion muß grundsätzlich folgende Aktionen durchführen:

- Berechnen des Speicherbedarfs für die eigenen Analyseergebnisse. Dieser Speicherbedarf ist abhängig von der Anzahl und der Kardinalität der Analyseergebnisse und von den Benutzerwünschen. Sollen für ein bestimmtes Protokoll keine Analysen durchgeführt werden, benötigt die Funktion auch keinen Speicherplatz für Ergebnisse.
- Berechnen der Analyseergebnisse. Die Analyseergebnisse werden in einer protokollspezifischen Struktur ab der als Parameter übergebenen Adresse für die Analysedaten gespeichert. Auf diese protokollspezifische Struktur wird beispielsweise von den weiter unten beschriebenen Verwaltungsfunktionen für den Datenexport (vgl. Abschnitt 5.3.4.3) zugegriffen.
- Berechnen lokaler Statistiken. Die lokalen Statistiken werden an die globalen Statistikfunktionen weitergeleitet (vgl. Abschnitt 5.3.3).

der Ethernet-Schnittstelle waren jedoch nur für einen vergleichbaren Compiler der Firma INMOS vorhanden.

- Erkennen eines eingebetteten Protokolls. Unabhängig davon, ob die Analysefunktion Analyseergebnisse berechnet, muß sie eine eventuell eingebettete Dateneinheit eines anderen Protokolls erkennen. Wie weiter oben dargestellt wurde, sind die dafür notwendigen Informationen protokollspezifisch (vgl. Abbildung 4.3).

- Aufrufen der Analysefunktion des erkannten Protokolls. Dies geschieht immer, wenn ein eingebettetes Protokoll identifiziert wurde. Ist kein weiteres Protokoll eingebettet oder handelt es sich um ein unbekanntes Protokoll, entfällt dieser Schritt.

  Als Parameter wird der Analysefunktion des eingebetteten Protokolls ein Zeiger auf die zu analysierenden Rohdaten übergeben. Dazu muß die aufrufende Analysefunktion einen Zeiger auf den Datenbereich der jeweiligen Dateneinheit berechnen[12]. Der zweite zu übergebene Parameter ist ein Zeiger auf den Speicherplatz für die Analyseergebnisse der aufgerufenen Analysefunktion. Dazu addiert die aufrufende Funktion ihren eigenen Speicherbedarf zu dem ihr übergebenen Zeiger. Mit anderen Worten, die aufgerufene Funktion legt ihre Ergebnisse unmittelbar hinter die Analyseergebnisse der aufrufenden Funktion im Speicherbereich ab.

  Die aufgerufene Funktion terminiert mit einem Rückgabewert, der ihren eigenen Speicherbedarf und den Speicherbedarf *iterativ* aufgerufener Analysefunktionen angibt.

- Berechnen des Gesamtspeicherbedarfs. Dieser Wert setzt sich zusammen aus dem eigenen Speicherbedarf für Analyseergebnisse (s.o.) und dem Rückgabewert der Analysefunktion des eingebetteten Protokolls.

- Terminieren und Übergeben des Gesamtspeicherbedarfs an die aufrufende Funktion.

In Abbildung 5.9 ist der Algorithmus noch einmal als Nassi-Schneiderman Diagramm dargestellt.

[12] In OSI-Terminologie: Es muß ein Zeiger auf die (n)-SDU der aktuellen (n)-PDU berechnet werden. Die (n)-SDU ist gerade die (n+1)-PDU, also das 'Rohdatum' für die folgende Analysefunktion.

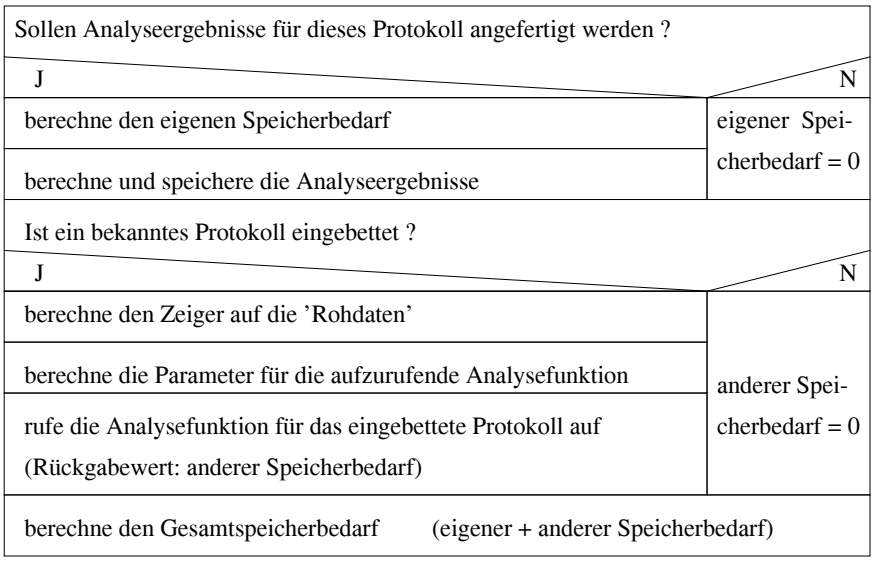

Sollen Analyseergebnisse für dieses Protokoll angefertigt werden ?
J
N
berechne den eigenen Speicherbedarf
berechne und speichere die Analyseergebnisse
eigener Speicherbedarf = 0
Ist ein bekanntes Protokoll eingebettet ?
J
N
berechne den Zeiger auf die 'Rohdaten'
berechne die Parameter für die aufzurufende Analysefunktion
rufe die Analysefunktion für das eingebettete Protokoll auf
(Rückgabewert: anderer Speicherbedarf)
anderer Speicherbedarf = 0
berechne den Gesamtspeicherbedarf (eigener + anderer Speicherbedarf)

Abbildung 5.9: 'Template' einer Analysefunktion

Der folgende 'Pseudo-C-Code' zeigt die Struktur des 'Templates'.

```
 1 #include "prototypen.h"
 2 #include "analyse_strukturen.h"
 3 extern int analysiere;

 4 int
 5 analysiere_protokoll (void *rohdaten, void *ergebnisspeicher)
 6 {
 7   int meine_datenlaenge, andere_datenlaenge, eingebettetes_protokoll;
 8   void *eingebettete_rohdaten, *anderer_ergebnisspeicher;

 9   MEINE_ANALYSE_STRUKTUR meine_rohdaten
10   = (MEINE_ANALYSE_STRUKTUR *) rohdaten;
11   MEINE_SPEICHER_STRUKTUR meine_ergebnisse
12   = (MEINE_SPEICHER_STRUKTUR *) ergebnisspeicher;

13   /* Eigene Ergebnisse berechnen... */
14   if (analysiere)
15     {
16       /* Ja, dieses Protokoll soll analysiert werden. */
17       meine_datenlaenge = sizeof (MEINE_SPEICHER_STRUKTUR);
18       meine_ergebnisse->wert1 = f1 (meine_rohdaten);
19       meine_ergebnisse->wert2 = f2 (meine_rohdaten);
20       /* weitere Berechnungen ... */
21       meine_ergebnisse->wert_n = f_n (meine_rohdaten);
22     }
23   else
24     {
25       /* Nein, dieses Protokoll soll nicht analysiert werden. */
26       meine_datenlaenge = 0;
```

```
27     }

28   /* Eingebettetes Protokoll erkennen und analysieren */
29   eingebettetes_protokoll = f_n + 1 (meine_rohdaten);
30   eingebettete_rohdaten = f_n + 2 (meine_rohdaten);
31   anderer_ergebnisspeicher = (char *) ergebnisspeicher
32     + meine_datenlaenge;

33   switch (eingebettetes_protokoll)
34     {
35     case PROTOKOLL1:
36       andere_datenlaenge = analysiere_protokoll1 (eingebettete_rohdaten,
37                                                   anderer_ergebnisspeicher);
38       break;
39     case PROTOKOLL2:
40       andere_datenlaenge = analysiere_protokoll2 (eingebettete_rohdaten,
41                                                   anderer_ergebnisspeicher);
42       break;
43       /* Andere bekannte Protokolle... */
44     case PROTOKOLL_N:
45       andere_datenlaenge = analysiere_protokoll_n (eingebettete_rohdaten,
46                                                    anderer_ergebnisspeicher);
47       break;
48     default:
49       andere_datenlaenge = 0;
50     }
51   return (meine_datenlaenge + andere_datenlaenge);
52 }
```

Der 'Pseudo-C-Code' verdeutlicht zwei Mechanismen, die unter anderem die Sprache 'C' für die Realisierung nahelegten. Es ist zum einen die elegante Zeigerarithmetik, mit deren Hilfe in den Zeilen 31–32 der Zeiger auf den Speicherplatz für die Analyseergebnisse der aufzurufenden Funktion berechnet wird. Der zweite Mechanismus wird in den Zeilen 9–12 dargestellt. Es wird angenommen, daß in einer externen Datei, die zur Übersetzungszeit geladen wird (Zeile 1 und 2), Typdefinitionen für die abstrakten Datentypen 'MEINE_ANALYSE_STRUKTUR' und 'MEINE_SPEICHER_STRUKTUR' vorhanden sind. Für den einfachen Zugriff auf die Rohdaten und den Ergebnisspeicher (Zeilen 17–21, 29–30) wird hier eine Typumwandlung auf Grundlage eines Zeigers vorgenommen. Danach ist ein strukturierter Zugriff auf einen vormals unstrukturierten Bereich ('void *' in Zeile 5) im Speicher möglich.

In der ebenfalls verfügbaren Programmiersprache OCCAM wäre ein derart einfacher und eleganter Zugriff auf den Speicher nicht möglich, da OCCAM keine Zeiger kennt.

Zu dem 'Pseudo-C-Code' ist noch anzumerken, daß sich hier die wesentlichen Anweisungen in den Zeilen 17–21 und 29–47 befinden. Einerseits ist hier die Analyse in abstrakter Form dargestellt[13] (17–21), andererseits das Erkennen und Analysieren eines eingebetteten Protokolls (29–47).

[13] Die Analyse wird hier hinter den Funktionsaufrufen $f_1 \cdots f_{n+2}$ verborgen.

5.3.2.2 Realisierung der Analysefunktion für 'Ethernet/- IEEE 803.2 Frames'

'Ethernet-Frames' (laut Xerox, Digital, Intel Spezifikation für Ethernet 2.0) unterscheiden sich von 'IEEE 802.3 Frames' in der Interpretation des mit 'Frame-Typ' bzw. 'Daten Länge' bezeichneten Abschnitts des 'Frames' (vgl. Abbildung 5.10). Während hier bei einem 'Ethernet-Frame' eine Typangabe die im Daten-Bereich eingekapselte Dateneinheit einer höheren Protokollebene identifiziert, steht bei 'IEEE 802.3 Frames' an dieser Stelle eine Längenangabe, die die Gesamtlänge des Datenfeldes (in diesem Fall die Länge des eingebetteten 'IEEE 802.2 Logical Link Control' (LLC)-'Frames') angibt. Der 'Frame' muß eine minimale Länge von 64 Byte haben, um Kollisionen sicher erkennen zu können. Alle 'Frames' mit einer Länge von weniger als 64 Byte werden als Fragmente einer Kollision angesehen und laut Spezifikation vernichtet (vgl. Black 1993, S.129). Durch die Festlegung, daß ein 'Ethernet-Frame' im 'Frame-Typ' keine Werte kleiner oder gleich 1518[14] haben darf, ist eine Unterscheidung der beiden 'Frames' möglich.

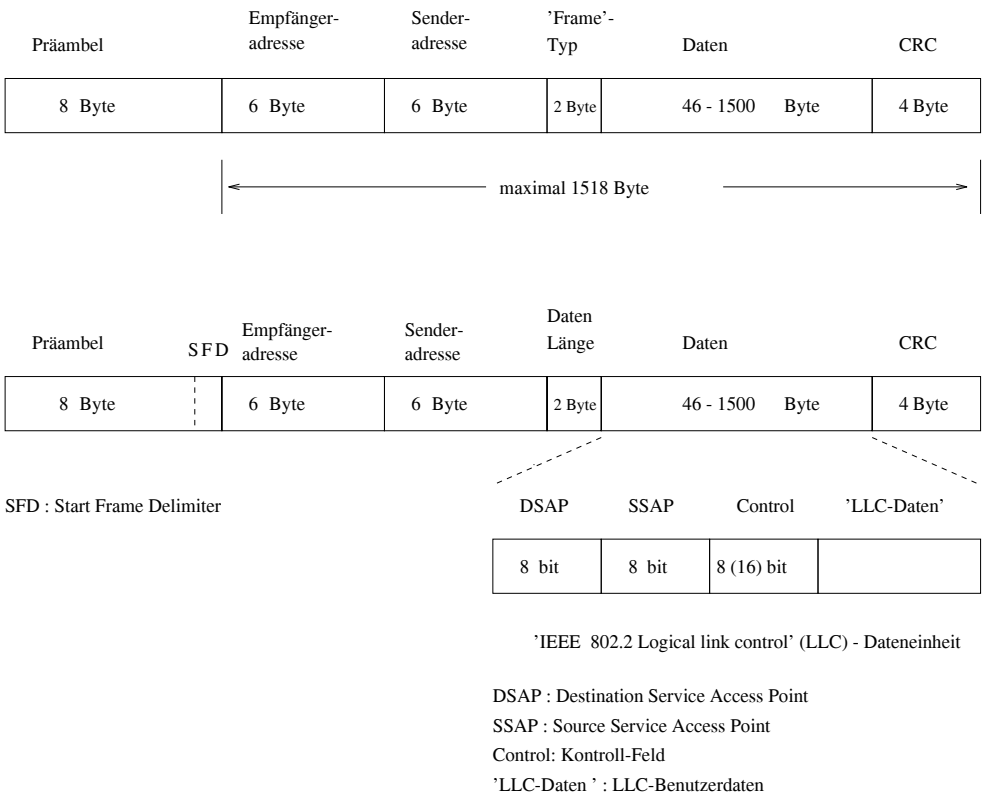


Abbildung 5.10: 'Ethernet- und IEEE 802.3 Frame' (vgl. Black 1993, S.124)

[14] Dies ist die maximale Länge eines 'IEEE 802.3 Frames' ohne Präambel.

Neben der Identifizierung des 'Frames' sollten bei einer anschließenden Analyse folgende Werte ermittelt werden:

1. Gesamtlänge des 'Frames', um gegebenenfalls die Brutto-Last ('Utilization') berechnen zu können.

2. Länge des Datenfeldes, um gegebenenfalls die Netto-Last bzw. den 'Protokolloverhead' berechnen zu können. Der 'Protokolloverhead' ist der Anteil an der Brutto-Last, der durch Übertragen von Informationen außerhalb des Datenfeldes verursacht wird. Der 'Protokolloverhead' wird demnach durch alle nicht zum Datenfeld gehörenden Teile des 'Frames' verursacht. Er ist in diesem Fall umgekehrt proportional zur Länge des 'Frames'.

   Die Länge des Datenfeldes kann auch an Analysefunktionen für eingebettete Protokolle (optional) übergeben werden. Dies bietet sich vor allem in Fällen an, in denen das eingebettete Protokoll nicht über eine explizite Angabe zur jeweiligen Länge der Dateneinheit verfügt. Man vermeidet so aufwendige Berechnungen der Längenangabe mittels anderer protokollspezifischer Angaben.

3. Erkennen des eingebetteten Protokolls. Bei 'Ethernet-Frames' ist dies trivial durch den Wert des 'Frame Type' Feldes möglich, wie in RFC 1700 spezifiziert. Aus diesem Grund werden 'Ethernet-Frames' aus der Sicht einer höheren Protokollschicht auch als 'selbst-identifizierende' 'Frames' bezeichnet. Bei 'IEEE 802.3-Frames' ist grundsätzlich ein 'LLC-Frame' laut 'IEEE 802.2'-Spezifikation eingebettet (vgl. Abbildung 5.10, 'LLC-Dateneinheit'). Steht für das erkannte Protokoll eine Analysefunktion zur Verfügung, kann sie an dieser Stelle aufgerufen werden.

4. Identifizieren des Senders und des Empfängers.

5.3.2.3 Realisierung der Analysefunktion für ARP/RARP Dateneinheiten

Das 'Address Resolution Protocol' (ARP) und das 'Reverse Address Resolution Protocol' (RARP) sind Protokolle der TCP/IP-Protokollfamilie. Die Hauptaufgabe dieser beiden Protokolle besteht darin, physikalische Adressen (in diesem Fall Ethernet-Adressen) auf 'Internet-Adressen' (IP-Adressen) und IP-Adressen auf physikalische Adressen abzubilden.

ARP und RARP benutzen das gleiche Nachrichtenformat[15] (vgl. Abbildung 5.11). Die Erkennung (s.o.) ist bei einer Einbettung dieser Protokolle in 'Ethernet-Frames' trivial. Laut RFC 1700 ist der 'Frame-Typ' für ARP auf 0806_{16} und für RARP auf 0835_{16} zu setzen (vgl. auch Abbildung 4.3).

Die Analyse von ARP und RARP muß folgendes leisten:

1. Im Feld 'Hardware-Typ' muß der Wert 1 (für Ethernet) verifiziert werden.

[15] Mit Nachrichtenformat ist die Struktur der Dateneinheit eines Protokolls gemeint.

2. Im Feld 'Protokoll-Typ' muß der Wert 0800_{16} (für IP) verifiziert werden.

3. Im Feld 'Funktion' muß bei ARP entweder der Wert 1 für ARP-Anfrage (ARP request) oder der Wert 2 für ARP-Antwort (ARP response) stehen. Bei RARP 3 (RARP request) oder 4 (RARP response). Diese Werte sind in Abhängigkeit vom 'Frame-Typ' (s.o.) zu überprüfen.

4. 'HLEN' gibt die Länge der physikalischen Adresse an und muß bei Ethernet den Wert 6 haben.

5. 'PLEN' gibt die Länge der Adresse des Protokolls einer höheren Protokollebene an. Für IP-Adressen muß hier der Wert 4 stehen (vgl. auch Abbildung 5.12, SA & EA).

Die Länge ist konstant 28 Byte. Sie braucht nicht berechnet zu werden. In ARP- und RARP-Dateneinheiten werden keine weiteren Protokolle eingebettet. Eine Erkennung ist deshalb nicht nötig.

Neben diesen invarianten Informationen muß die Analyse bei einer 'ARP-Anfrage' die physikalische Adresse (hier Ethernet-Adresse) des Senders ('physikalische Senderadresse') und die IP-Adresse des Senders ('IP-Senderadresse') gewinnen. Bei einer 'ARP-Antwort' läßt sich zusätzlich auch das Adreßpaar des Empfängers gewinnen. Bei einer RARP-Antwort kann das Adreßpaar des Empfängers gewonnen werden (vgl. [Com91, S. 80ff]).

Die variablen Teile des (R)ARP-Protokolls (hier die Adreßpaare für Sender und Empfänger) sind von besonderer Bedeutung für die Netzanalyse. Einerseits kann durch das schlichte Anzeigen einer ARP-Tabelle -mit den Zuordnungen der Adressen zueinander- geprüft werden, ob beispielsweise eine Adresse mehrmals vergeben wurde[16]. Andererseits ermöglichen ARP-Tabellen zusammen mit 'Domain Name System' (DNS)-Informationen die Auflösung eines symbolischen Namens bis zur physikalischen Adresse. Das bedeutet eine wesentliche Erleichterung für den Benutzer, der bei der Eingabe von Rechneradressen i.d.R. symbolische Namen bevorzugt.

[16] Eine Überprüfung auf mehrfache Adreßvergabe kann auch automatisch erfolgen.

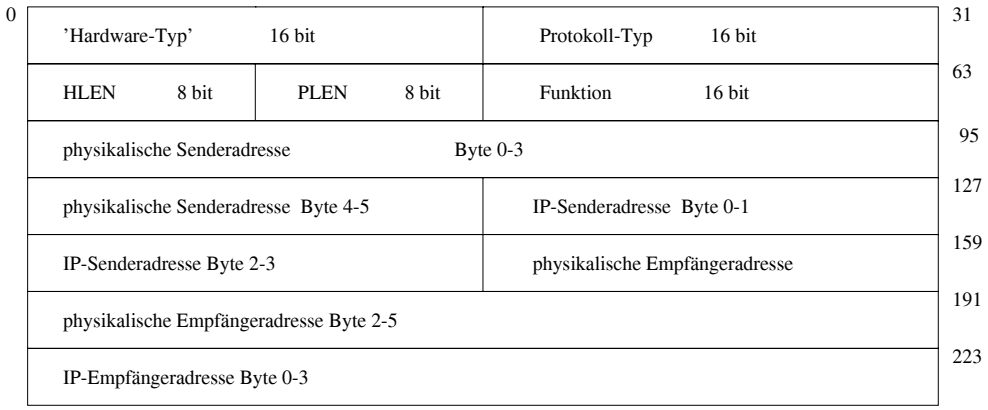


Abbildung 5.11: (R)ARP-Dateneinheit (vgl. [Com91, S. 80]

5.3.2.4 Realisierung der Analysefunktion für Datagramme

IP-Dateneinheiten (Datagramme) sind die Grundlage aller Datenübertragungen innerhalb der TCP/IP-Protokollfamilie. Datagramme und ARP/RARP-Dateneinheiten sind die einzigen Dateneinheiten der TCP/IP-Protokollfamilie, die unmittelbar in 'Ethernet-Frames' eingebettet werden. Alle anderen Dateneinheiten dieser Protokollfamilie werden in Datagramme eingebettet. Identifiziert wird ein Datagramm durch den Eintrag 0800_{16} im Typ-Feld des 'Ethernet-Frame'.

Das in Abbildung 5.12 dargestellte Nachrichtenformat hat nur einen konstanten Eintrag. Das Feld VER (Version) gibt die aktuelle Versionsnummer für Datagramme laut RFC 791 an und muß den Wert vier (4) haben. Alle anderen Einträge haben variable Werte mit folgender Bedeutung:

1. Das Feld KL (Kopflänge) gibt die Länge des Datagrammkopfes in 32bit-Worten an. Der Datagrammkopf hat aufgrund der variablen Anzahl von Optionen (s.u.) keine feste Länge. Er umfaßt alle Felder bis einschließlich des Füllfeldes (FÜLL). Das Füllfeld besteht ausschließlich aus Nullen (0) und erweitert die letzte Option gegebenenfalls auf eine Länge von 32bit, so daß die anschließenden Daten (IP-Daten) auf 32bit innerhalb des Datagramms ausgerichtet sind. Ein typischer Eintrag für KL ist fünf (5). Ein 20 Byte langer IP-Kopf ((KL = 5) * 4) hat weder Optionen noch ein Füllfeld.

2. Der Diensttyp (DT) ist seinerseits unterstrukturiert. Das Vorrangfeld (VR) enthält einen Wert von null (0) bis sieben (7). Dieser Wert wird von den meisten Vermittlungsrechnern ignoriert. Konzeptionell sollte ein Datagramm mit höherem Wert in VR vor einem Datagramm mit niedrigerem Wert in VR vermittelt werden. Die Bits V, D und Z sollten im gegenseitigen Ausschluß gesetzt sein. Sie geben an, ob ein Datagramm mit geringer Verzögerung (V=1), mit

hohem Durchsatz (D=1) oder mit hoher Zuverlässigkeit vermittelt werden soll. Für einige anwendungsorientierte Protokolle gibt es Vorschläge, wie diese Bits in Abhängigkeit vom Typ der eingebetteten Dateneinheiten zu setzen sind. Die beiden letzten Bits des DT-Feldes werden nicht benutzt. Sie sollten den Wert null (0) haben.

3. Das Feld Gesamtlänge (GL) beinhaltet die Länge des Datagramms einschließlich des Kopfes. Die Angabe ist in Byte (8bit). Daraus ergibt sich, daß die Gesamtlänge maximal 65535 Byte beträgt. Bei minimaler Kopflänge (20 Byte, s.o.) ist die maximale Länge der IP-Daten 65515 Byte. Die Länge der IP-Daten kann durch die Formel : IP-Daten-Länge = GL - KL * 4 berechnet werden.

4. Die Felder Identifikation (ID), Flaggen (FL) und Fragment Position (FP) sind nötig, um das richtige Zusammensetzen fragmentierter Datagramme sicherzustellen. Ist die Länge eines Datagramms länger als die maximale Länge der auf einmal übertragbaren Daten[17] (maximal transfer unit = MTU), so muß das Datagramm in mehrere Fragmente mit einer Länge kleiner oder gleich MTU zerlegt werden. Im Falle einer Fragmentierung bekommen alle Fragmente eines Datagramms die gleiche ID, damit beim Zusammensetzen der Fragmente nach der Übertragung nicht versehentlich zwei Fragmente verschiedener Datagramme zusammengefügt werden. Fragment Position gibt an, an welcher Stelle im fragmentierten Datagramm die Informationen dieser Dateneinheit liegen. Der Wert von FP wird hierzu mit acht (8) multipliziert, um die tatsächliche Position der Daten des Fragments innerhalb des Datagramms zu ermitteln (vgl. [Com91, S. 97]).

   Die drei Bits mit der Bezeichnung Flaggen geben an, ob ein Datagramm fragmentiert werden darf (Bit1 = 0) und ob weitere Fragmente zu erwarten sind (Bit2 = 1). Das bedeutet, alle Fragmente eines Datagramms, mit Ausnahme des letzten, müssen das zweite Bit setzen. Das letzte Fragment muß das zweite Bit der Flaggen zurücksetzen. Über die Gesamtlänge (GS), der Fragmentposition und den Flaggen kann bestimmt werden, ob ein Datagramm vollständig wieder zusammengesetzt wurde. Das Bit0 wird nicht benutzt und muß immer den Wert null (0) haben (vgl. RFC 791, S. 13).

5. Das Feld Lebenszeit (LZ) gibt an, wieviele Sekunden ein Datagramm in einem Internetzwerk vermittelt werden darf (vgl. [Com91, S. 99]). Jeder Vermittlungsrechner ist angewiesen, dieses Wert um mindestens eins (1) zu vermindern und das Datagramm zu vernichten, falls der Wert danach null (0) ist.

6. Das Feld Protokoll (PR) identifiziert analog zum Ethernet-Typ-Feld das eingebettete Protokoll. Aus der Sicht eines höheren Protokolls ist somit auch IP ein 'sich selbst identifizierendes' Protokoll.

7. Das Feld Kopfprüfsumme (KP) enthält die Prüfsumme über den gesamten Kopf[18].

[17] Bei 'Ethernet-Frames' ist die MTU 1500 Byte.

[18] Zur Berechnung nimmt man an, daß die Prüfsumme den Wert 0000_{16} hat.

8. Die Felder Senderadresse (SA) und Empfängeradresse (EA) enthalten IP-Nummern.

9. Die (optionalen) Optionen sind komplex strukturiert. Sie ermöglichen die Funktionalisierung von Datagrammen für einfache Netzmanagement-Aufgaben[19].

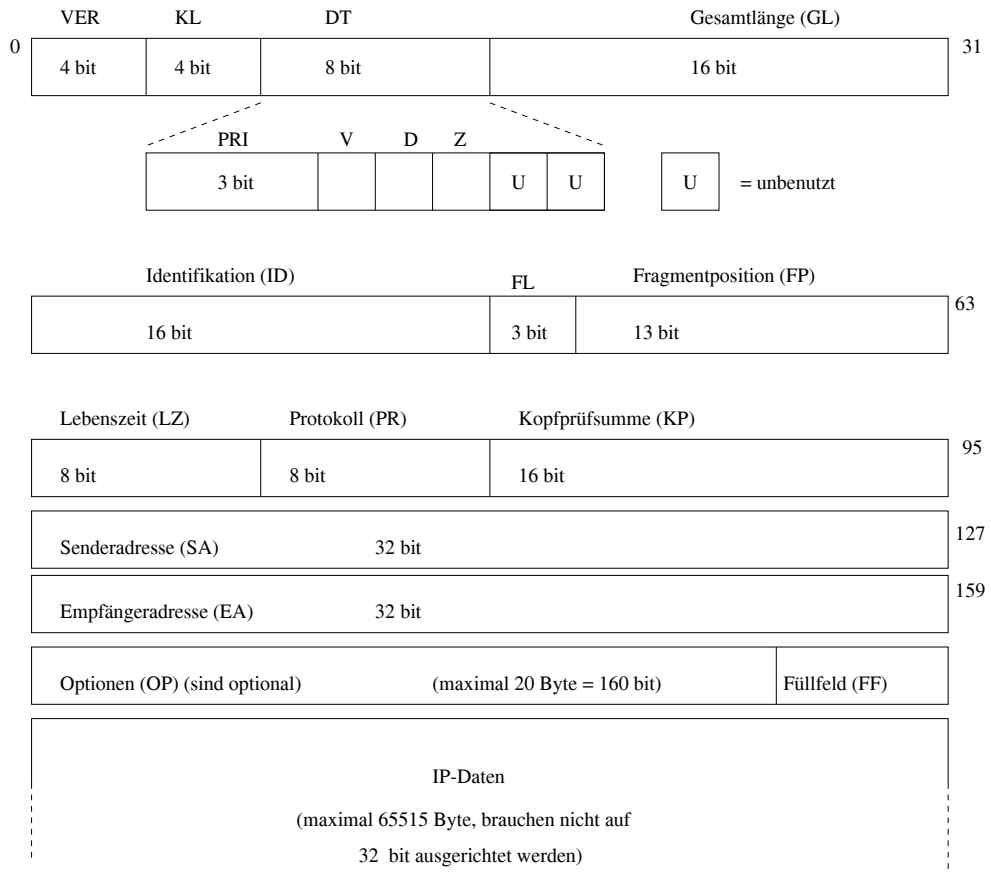


Abbildung 5.12: Datagramm (IP-Dateneinheit) (vgl. RFC 791, S. 11)

Format der IP-Optionen

Die Anzahl und Länge einzelner Optionen ist variabel. Optionen sind strukturiert nach *Klasse* und *Optionsnummer*.

Bei einer Analyse kann geprüft werden, ob die Bits für die Klasse der Option richtig gesetzt sind. Von den vier möglichen Werten sind nur die Werte null (0) für

[19] Beispielsweise kann durch die 'strikte Vorgabe' für einen zu verwendenden Pfad geprüft werden, ob bestimmte Vermittlungsrechner erreichbar sind.

'Datagramm oder Netzwerk-Kontrolle' sowie zwei (2) für 'Fehlersuche und Messungen' erlaubt. Die Werte eins (1) und drei (3) sind reserviert für künftige Erweiterungen. Neben der Grobeinteilung in die Klassen null oder zwei muß verifiziert werden, ob die Optionsnummer bekannt ist und mit der Optionsklasse verträglich ist. In Tabelle 5.1 sind alle möglichen Kombinationen aufgeführt.

Optionsklasse	Optionsnummer	Länge	Beschreibung
0	0	-	Ende der Optionsliste. Immer dann, wenn die Optionen nicht mit dem IP-Kopf enden.
0	1	-	Ausrichten von Byte in einer Optionsliste.
0	2	11	Für militärische Anwendungen.
0	3	variabel	Vorgaben für einen zu verwendenden Pfad.
0	7	variabel	Pfad aufzeichnen, den das Datagramm vermittelt wird.
0	8	4	Historisch, wird nicht mehr benutzt.
0	9	variabel	Strikte Vorgabe für einen zu verwendenden Pfad.
2	4	variabel	Zeitstempel der Vermittlungsrechner anfordern.

Tabelle 5.1: IP-Optionen

Es fällt auf, daß bei weitem nicht alle 32 möglichen Optionsnummern in Kombination mit den beiden möglichen Optionsklassen zulässig sind. Außerdem ist von den acht (8) möglichen Kombinationen nur die Optionsnummer vier (4) mit der Optionsklasse 'Fehlersuche und Messungen' (2) verträglich.

5.3.2.5 Realisierung der Analysefunktion für das 'Internet Control Message Protocol'

Das 'Internet Control Message Protocol' (ICMP) wird in der TCP/IP-Protokoll-Familie benutzt, um Fehlermeldungen und Kontrollnachrichten zu versenden. Das Format einer ICMP-Nachricht wird durch die beiden Felder 'TYP' und 'CODE' bestimmt. Das allgemeine Nachrichten-Format einer ICMP-Dateneinheit ist in Abbildung 5.13 abgebildet. ICMP-Dateneinheiten werden in Datagramme eingebettet. Der Wert 'Protokoll' im IP-Kopf wird zur Identifizierung von ICMP-Dateneinheiten auf den Wert

0001_{16} gesetzt.

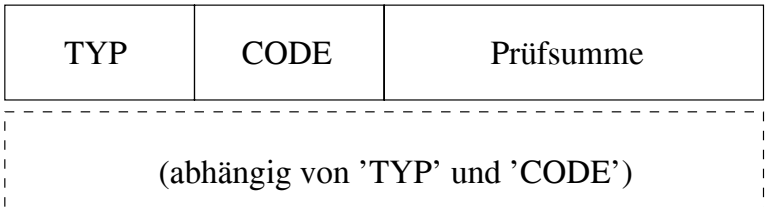


Abbildung 5.13: ICMP-Dateneinheit

Die Hauptaufgabe bei der Analyse einer ICMP-Dateneinheit besteht darin, die vielfältigen Kombinationsmöglichkeiten aus 'TYP' und 'CODE' in eine Klartextmeldung umzuwandeln. Diese Meldung ist der wesentliche Teil bei der Protokollierung von ICMP-Dateneinheiten (vgl.[Com91, S.127–136], RFC 792, S. 4–18). Außerdem wird die Länge der Dateneinheit bestimmt.

5.3.2.6 Realisierung der Analysefunktion für das 'User Datagram Protocol'

Das 'User Datagram Protocol' (UDP) erweitert das 'Internet Protocol' (IP) im wesentlichen um die Angabe eines 'Quell-Ports' und eines 'Senken-Ports'. Wird hier eine gemäß RFC-1700 standardisierte Nummer eingetragen, kann der Empfänger hiermit die Anwendung identifizieren, die die Daten versendet hat bzw. für welche Anwendung die Daten bestimmt sind.

Die Struktur von UDP-Dateneinheiten ist entsprechend simpel (vgl. Abbildung 5.14).

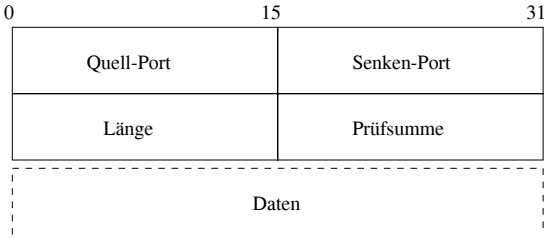


Abbildung 5.14: UDP-Dateneinheit

Die Analysefunktion für UDP-Dateneinheiten muß neben den üblichen Berechnungen zur Länge nur mit Hilfe einer Tabelle das eingebettete Protokoll anhand der 'Portnummern' identifizieren. Danach könnte die entsprechende Analysefunktion mit einem Zeiger auf den UDP-Benutzerdatenbereich aufgerufen werden.

5.3.2.7 Realisierung einer Analysefunktion für das 'Transmission Control Protocol'

Das 'Transmission Control Protocol' (TCP) stellt einen zuverlässigen verbindungsorientierten Dienst zur Verfügung. Es bietet Flußkontrollmechanismen und gestattet es den beiden Kommunikationspartnern festzulegen, wieviele Daten ohne Bestätigung 'vorausgesendet' werden dürfen[20].

Außerdem ermöglicht TCP eine Unterscheidung zwischen 'normalen' Daten und 'Vorrangdaten', die unmittelbar zu bearbeiten sind. Bei den 'normalen' Daten sorgt TCP außerdem dafür, daß diese in der richtigen Reihenfolge an den Empfänger weitergeleitet werden.

Aus dieser Funktionsvielfalt ergibt sich die komplexe Struktur der TCP-Dateneinheiten (vgl. Abbildung 5.15).

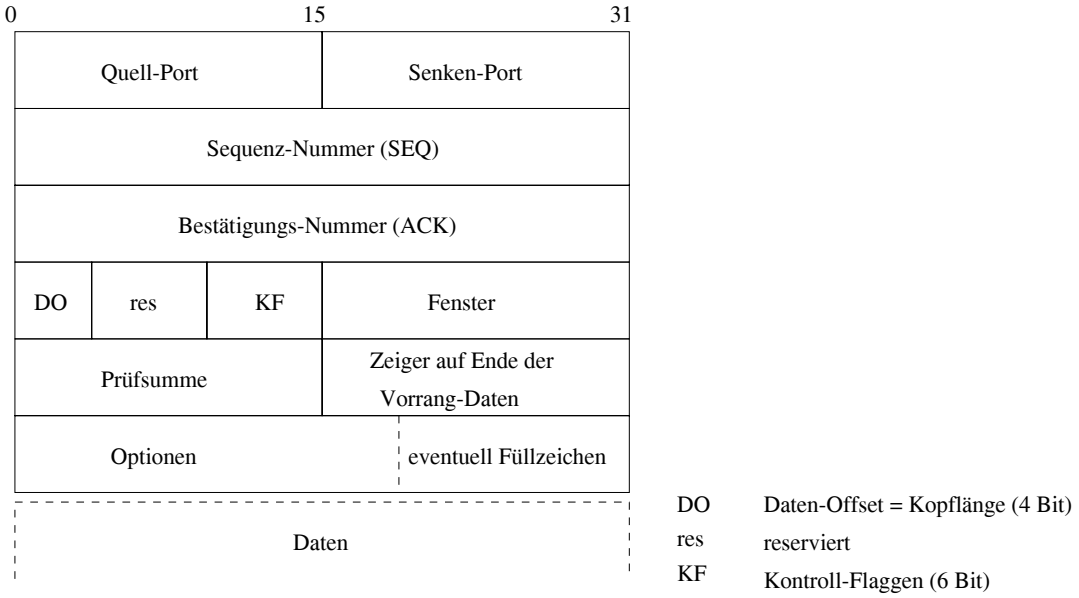


Abbildung 5.15: TCP-Dateneinheit

Die Analysefunktion für TCP-Dateneinheiten berechnet die folgenden Werte:

- Längenangaben zur Gesamtlänge, Datenlänge und Optionenlänge,
- Wert der Sequenznummer (SEQ) und der Bestätigungsnummer (ACK),
- 'Quellen-' und 'Senken-Port'[21] und

[20] Dieser Mechanismus dient dazu, den Betriebsmittelbedarf für die Empfangspuffer der Verbindung zu kontrollieren.

[21] Die TCP-Portnummern identifizieren genau wie die UDP-Portnummern eine Anwendung bzw. ein eingebettetes Protokoll.

- die Kontroll-Bits.

Diese Daten können wie bei den anderen Analysefunktionen zum Berechnen lokaler Statistiken benutzt werden oder sie werden verwendet, um Verbindungen zu protokollieren.

Verbindungsorientierte Protokolle und die Paketparallelität

Das Analysieren verbindungsorientierter Protokolle (z.B. TCP) nach dem Konzept der Paketparallelität ist nur scheinbar ein Widerspruch.

Vielmehr ist das Analysieren von verbindungslosen Protokollen mit dem Konzept der Paketparallelität besonders leicht, eine Analyse von verbindungsorientierten Protokollen nur u.U. aufwendiger. Bei den oben dargestellten Berechnungen unterscheidet sich die Analyse überhaupt nicht. Dies liegt daran, daß zur Zeit nur auf Werte zugegriffen wird, die explizit in den Dateneinheiten mitgeführt werden.

Soll bei Berechnungen auf *Zustände einer Verbindung* eingegangen werden, muß für das jeweilige Protokoll natürlich auch für jede Verbindung ein Protokollautomat geführt werden. Das Führen des Protokollautomaten (Zustandsänderungen protokollieren) ist wiederum protokollspezifisch. Das *Aufteilen* der Verbindungen auf die Paketträger muß allerdings durch Kooperation der Paketträger untereinander erfolgen. Die dafür vorgesehenen Funktionen wurden in Abschnitt 4.5.2 entworfen und werden anhand von TCP konkretisiert.

Erkennen und Verwalten von Verbindungen

Das Erkennen und Aufteilen der Verbindungen wird hier kurz anhand von TCP diskutiert. Im Prototypen ist dies bisher nicht realisiert.

Jeder Paketträger, der eine Dateneinheit eines verbindungsorientierten Protokolls empfängt, muß folgende Aktionen durchführen:

- Erkennen der Verbindung. Bei TCP ist eine Verbindung eindeutig durch das Adreßpaar von Sender und Empfänger (im IP-Datagramm) und den Portnummern identifizierbar.
- Feststellen, ob der Paketträger diese Verbindung verwaltet. Handelt es sich um die erste Dateneinheit eines Verbindungsaufbaus, ist der Paketträger, der diese Dateneinheit auswertet, auch für die Verwaltung der Verbindung zuständig. Andere Dateneinheiten gibt er (z.B. mit einer 'Multicast'-Nachricht) an die anderen Paketträger weiter.
- Empfängt er keine Bestätigung, daß ein anderer für eine weitergeleitete Dateneinheit zuständig ist, ist er auch für diese Verbindung zuständig[22].

[22] Ein solcher Fall tritt immer dann ein, wenn eine Dateneinheit empfangen wird, die zu einer Verbindung gehört, die vor Beginn der Messung aufgebaut wurde.

- Optional können die Paketträger bei geringer Auslastung des Meßwerkzeuges Tabellen mit Verbindungsdaten austauschen. Dadurch läßt sich für viele Verbindungen der jeweils zuständige Paketträger auch direkt adressieren, falls eine Dateneinheit weitergereicht werden muß (s.o.).

Es wurden keine Verbindungsmanagement-Funktionen für den Prototypen realisiert.

5.3.3 Realisierung der globale Statistikfunktionen

Der Algorithmus für die Berechnung der globalen Statistik ist in Abbildung 5.16 dargestellt. Berechnet werden beim gegenwärtigen Prototyp nur Daten zum Durchsatz auf den verschiedenen Protokollebenen in Megabit pro Sekunde. Denkbar sind weitere Statistiken wie in Abschnitt 2.1 angedeutet.

Die beim Entwurf (vgl. Abschnitt 4.4.3) genannte Periode von einer Sekunde ist optimal für die Berechnung der Auslastung in MBit/s. Die lokalen Statistiken werden in Byte/s geführt, so daß für die Vorbereitung der Statistik für den 'Hostrechner' nur eine Umrechnung in Bit erforderlich ist.

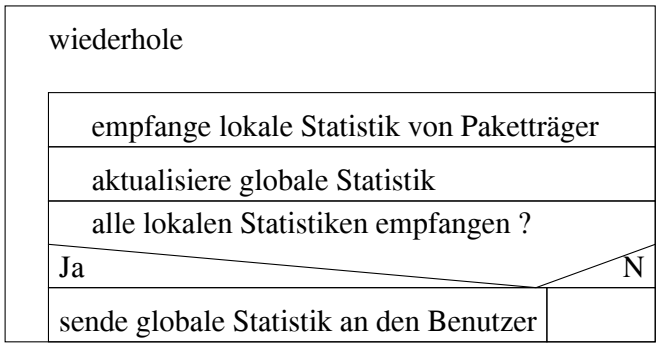


Abbildung 5.16: Berechnung der globalen Statistiken

5.3.4 Realisierung der Funktionen für den 'Hostrechner'

Im folgenden werden die Realisierungen der essentiellen Funktionen des 'Hostrechners' dargestellt. Die Darstellung orientiert sich am Entwurf aus Abschnitt 4.4.4.

5.3.4.1 Realisierung von Benutzereingaben

Benutzer können Anweisungen über eine *Kommandozeile* eingeben oder über *Menüs* der Programmfenster (mit Hilfe der 'Maus' auf der graphischen Benutzerschnittstelle). Eingaben über eine Kommandozeile sind für den geübten Benutzer häufig schneller und effizienter zu machen als über eine (eventuell) mehrstufige Menühierarchie.

Die Eingabe über Menüs und 'Dialogfelder' der graphischen Benutzerschnittstelle bietet sich in Fällen an, in denen aus einer Vielzahl von vorgegebenen Wahlmöglichkeiten selektiert werden muß. Dies ist beispielsweise der Fall, wenn der Benutzer angeben soll, welche Meßdaten zu erheben sind.

Kommandozeile

Über die Kommandozeile können über 'Schlüsselwörter' Kommandos eingegeben, korrigiert und verändert werden. Dazu ist es notwendig, die Schlüsselwörter als solche zu erkennen und die Einhaltung syntaktischer Regeln bei der Kombination der Schlüsselwörter mit anderen Symbolen[23] zu überprüfen.

Schlüsselwörter und andere Symbole werden aus dem Zeichenstrom durch einen 'Scanner' erkannt. Die reservierten Schlüsselwörter und Regeln, wie die anderen zu erkennenden Symbole aufgebaut sein müssen, wurden mit Hilfe eines 'Lex-Skriptes' definiert. Das unter Abschnitt 3.2.2.1 erwähnte Werkzeug 'Lex' generierte daraus den Quellcode einer Funktion, die Schlüsselwörter in Zeichenketten erkennt. Einem erkannten Schlüsselwort wird eine eindeutige Nummer zugeordnet. Die anschließende syntaktische Überprüfung der Eingabe wird dadurch beschleunigt. Anstelle der Schlüsselwöter müssen nur noch 'Token'(-Nummern) verarbeitet werden.

Für die syntaktische Überprüfung der Eingabe wurde ein 'Yacc-Skript'[24] geschrieben. Darin ist festgelegt, wie die verschiedenen Schlüsselwörter und Symbole angeordnet werden dürfen und welche Umwandlungsfunktionen beim Erkennen einer gültigen Benutzereingabe ausgeführt werden sollen. Das Werkzeug 'Bison' (vgl. Abschnitt 3.2.2.1) generiert aus diesem Skript ebenfalls eine Funktion. Erweiterungen um neue Kommandos sind daher sehr leicht durch Ergänzen von Regeln in den beiden Skripten möglich.

Durch die Trennung der lexikalischen und syntaktischen Überprüfung der Eingabe kann man die beiden Module unabhängig voneinander austauschen oder erweitern. Beispielsweise kann man die Schlüsselwörter gegen andere austauschen[25].

Durch die Verwendung der Werkzeuge 'Lex' und 'Yacc' werden Erweiterungen deutlich vereinfacht. Der Programmierer schreibt Regeln, nach denen die Überprüfung der Eingabe vorgenommen wird. Er muß jedoch nicht spezifizieren, *wie* diese Regeln zu überprüfen sind. Bei einer 'handgeschriebenen' Überprüfung würde dieser Aspekt dominieren.[26]

Die Abfrage der Benutzereingabe übernimmt ein Dialogelement[27] der graphischen Benutzerschnittstelle. Dadurch sind sehr komfortable Editiermöglichkeiten gegeben.

[23] z.B. Zahlen, IP-Adressen etc.

[24] Eine ausführliche Beschreibung der Werkzeuge 'Lex' und 'Yacc' wird in [LMB92] gegeben. Eine praktische Einführung in die Benutzung der Werkzeuge findet man beispielsweise in [SF85]. Der detaillierte Entwurf und die Realisierung vergleichbarer Werkzeuge wird in [Hol90] beschrieben.

[25] Durch einen Austausch der Schlüsselwörter könnte man das Werkzeug an andere Sprachen anpassen, beispielsweise an Französisch oder Polnisch.

[26] Für einen ausführlichen Vergleich 'handgeschriebener' Funktionen mit 'werkzeuggenerierten' vgl. [Job92].

[27] Diese sogenannten 'Editier-Fenster' werden als Standardfenster von OS/2 bereitgestellt.

Beispielsweise kann der Benutzer in seinem Kommando weitere Teile einfügen, löschen und automatisch formatieren lassen. Das Dialogelement liefert nach der Eingabe eine Zeichenkette mit der Benutzereingabe und initiiert die lexikalische und syntaktische Überprüfung.

Bei Eingabe eines korrekten Kommandos wird die entsprechende Aktion ausgelöst und das Dialogelement löscht die Eingabezeilen. Ein weiteres Kommando kann eingetippt werden. Eine falsche Eingabe (lexikalisch oder syntaktisch falsch) wird durch einen Warnton und eine Fehlermeldung im Ausgabefenster (s.u.) angezeigt. In diesem Fall werden die Eingabezeilen nicht gelöscht, um dem Benutzer eine Korrektur zu ermöglichen.

Dialogfelder

Diese Eingabeformulare werden auf der graphischen Benutzeroberfläche dargestellt. Der Benutzer kann mit der 'Maus' schnell und effizient umfangreiche Auswahlen auf diesen Formularen treffen. Die vom Benutzer gemachten Angaben können dann wiederum in entsprechende Aktionen umgewandelt werden.

Die Realisierung der Eingabeformulare ist relativ einfach. Mit entsprechenden Entwicklungswerkzeugen kann das Aussehen der Formulare graphisch-interaktiv bestimmt werden. Die Verknüpfung mit dem Programm ist jedoch sehr aufwendig, da für jedes Formular eine Funktion zur Reaktion auf die Benutzereingabe entwickelt werden muß.

Abschließend ist zu der Benutzereingabe anzumerken, daß die Kombination von Kommandoeingabe und Eingabe über Dialogfelder eine sehr effiziente Schnittstelle für den Benutzer darstellt. Die Programmierung ist jedoch entsprechend aufwendig.

5.3.4.2 Realisierung von 'Textfenstern'

Die Realisierung von Fenstern zum Anzeigen von Text ist aufgrund der Anforderungen beim Entwurf (vgl. Abschnitt 4.4.4) deutlich aufwendiger als die Realisierung der Dialogfelder. Die Darstellung der vergleichsweise statischen Dialogfelder kann der graphischen Benutzeroberfläche überlassen werden. Die Darstellung von Text und das 'Rollen' von Textausschnitten im Fenster muß aus zwei Gründen vom Programmierer selbst realisiert werden:

1. Die Fenstergröße und somit der sichtbare Textausschnitt kann vom Benutzer verändert werden.

2. Der Text ist nicht statisch, sondern wird dynamisch durch neuen Text ersetzt.

Es wird ein Ringpuffer für den Text benötigt, der eine bestimmte Anzahl von Textzeilen aufnehmen kann. Der im Fenster sichtbare Textausschnitt entspricht einem Abschnitt dieses Ringpuffers, der vom Benutzer vergrößert oder verkleinert werden kann.[28] Durch Betätigen eines 'Rollbalkens' an der Seite des Textfensters kann der

[28] Der Benutzer kann durch Verändern der Fenstergröße wählen, wieviele Zeilen Text dargestellt werden.

Benutzer den sichtbaren Textauschnitt (vertikal) verschieben. Dadurch ist es möglich, Textzeilen wieder sichtbar zu machen, die aus dem Fenster herausgerollt wurden.

Die eigentliche Schwierigkeit besteht in der Berechnung des gerade sichtbaren Textausschnittes. Der sichtbare Textausschnitt wird von den vier Faktoren

- Fenstergröße,
- Position des 'Rollbalkens',
- Schriftgröße und
- Überdeckung durch andere Fenster

bestimmt. Die Überdeckung durch andere Fenster schränkt den tatsächlich sichtbaren Bereich (determiniert durch die Fenstergröße) ein. Aus dem sichtbaren Bereich und der Schriftgröße kann man die Anzahl der sichtbaren Textzeilen berechnen. Die Position des 'Rollbalkens' kann man sich als Zeiger in den Ringpuffer vorstellen, der auf die Startposition zum Auslesen der Textzeilen zeigt.

Die optionale Ausgabe des Textes in eine Datei ist leicht realisierbar. Überprüfungen des gewählten Dateinamens und Betriebssystemaufrufe zum Öffnen, Schreiben und Schließen der Datei sind einfach realisierbar.

Die Ausgabe von Text in Fenstern wurde als abstrakter Datentyp implementiert. Der Ringpuffer, die Funktionen für obige Berechnungen und die Funktionen zur Ausgabe in Fenstern und Dateien sind vollständig gekapselt. Weitere 'Textfenster' können bei Bedarf erstellt werden. Die Benutzung ist trivial. Der Aufruf der Funktion 'wprintf()' schreibt in das als Parameter zu übergebene Fenster. Die weiteren Parameter und deren Bedeutung sind identisch mit der in 'C' bekannten 'printf()'-Funktion.

5.3.4.3 Realisierung eines benutzernahen Datenexports

Mit Hilfe von Umwandlungsfunktionen werden die Meßdaten in ein lesbares Format 'verschriftlicht' und anschließend in eine Datei geschrieben. Das 'Verschriftlichen'[29] der Analysedaten geschieht durch das Markieren jeder Information mit einer Beschreibung, die das folgende Datum identifiziert. Dadurch kann ein Benutzer leicht die Bedeutung des jeweiligen Datums identifizieren. Darüber hinaus wird beim Export darauf geachtet, daß die Zeilenlänge 80 Zeichen nicht überschreitet. Eine solche Datei kann dann mit jedem gängigen Editor angezeigt und auf jedem Drucker ausgedruckt werden. Das folgende Beispiel zeigt einen Auszug aus einer Datei mit benutzernahen Protokollinformationen.

```
# ETH_ANALYSER :
# MODUS protokollieren  Gewaehlte Protokolle : Ethernet ARP/RARP IP UDP TCP ICMP
ZEIT [ms] =   1168, EVENT Nr. =     91
ETH: laenge =  142, SA = 00:00:0c:0c:22:50, RA = 08:00:20:20:ab:39, IP
IP: laenge =  124, Opt. =   0, SA =     134.100.9.61, RA =     134.100.15.1
IP: Routine, Normal_Delay, Normal_Throughput, Normal_Reliability
IP: Protokoll = UDP,
UDP: laenge = 104, Source Port = 2049, Dest. Port = 1020
```

[29] Es handelt sich hierbei im wesentlichen um den Vorgang des *Protokollierens*.

```
ZEIT [ms] =   6138, EVENT Nr. =    358
ETH: laenge =   82, SA = 00:00:0c:0c:22:50, RA = 01:00:5e:00:00:05, IP
IP: laenge =   64, Opt. =   0, SA =   134.100.15.250, RA =        224.0.0.5
IP: Internetwork Control, Normal_Delay, Normal_Throughput, Normal_Reliability
IP: Protokoll = OSPFIGP,

ZEIT [ms] =   6146, EVENT Nr. =    359
ETH: laenge =   64, SA = 00:00:0c:0c:22:50, RA = 09:00:2b:01:00:01, UNBEKANNT

ZEIT [ms] =   6248, EVENT Nr. =    360
ETH: laenge =   64, SA = 00:80:d3:a0:72:38, RA = 00:00:0c:0c:22:50, IP
IP: laenge =   40, Opt. =   0, SA =    134.100.15.82, RA =     134.100.9.61
IP: Routine, Normal_Delay, Normal_Throughput, Normal_Reliability
IP: Protokoll = TCP,
TCP: laenge =   20, Opt. =   0, SP = 1713, DP = 43049
TCP: SEQ 318280796, ACK 0
TCP: FLAGS = FIN RST
```

5.3.4.4 Realisierung eines maschinennahen Datenexports

Anforderung an diese Unterstützungsfunktion ist es, die Analysedaten so aufzubereiten, daß ein externes Werkzeug effizient darauf zugreifen kann. Da es sich bei diesen Werkzeugen in der Regel um bekannte 'Textprozessoren'[30] wie AWK, SED oder PERL handelt (vgl. z.B. [Hol92]), wird auch in diesem Fall eine ASCII-Datei erzeugt.

Im Gegensatz zu einer benutzernahen Aufbereitung braucht man hier keine Rücksicht auf Zeilenlängen zu nehmen. Die redundanten Markierungen (s.o.) werden durch eine einmalige Beschreibung am Anfang der Datei in einer Präambel ersetzt. In dieser Präambel kann das externe Werkzeug die Bedeutung und die Reihenfolgen der abgespeicherten Daten erkennen. Dadurch ist eine redundanzarme Darstellung möglich, die auch bei großen Datenmengen eine kompakte Speicherung ermöglicht. Das folgende Beispiel zeigt einen maschinennahen Datenexport. Die erwähnte Präambel ist durch ein Kommentarsymbol (#) gekennzeichnet.

```
# ETH_ANALYSER :
# MODUS Messdaten erfassen  Gewaehlte Protokolle : IP UDP TCP ICMP
# IP Sender Empfaenger Laenge Protokoll Optionenlaenge Datenlaenge
# UDP Quell-Port Senken-Port Laenge Datenlaenge
# TCP Quell-Port Senken-Port Laenge Optionenlaenge Datenlaenge 'Code-Bits'
# ICMP Type 'Code' Laenge

ZEIT    472
IP 134.100.15.82 134.100.9.61   40 TCP     0   20
TCP   1713  43049   20    0    0 PSH ACK

ZEIT    649
IP 134.100.15.82 134.100.9.61   41 TCP     0   21
TCP   1713  43049   21    0    1 SYN RST PSH ACK
```

[30] Diese Textverarbeitungen laufen im 'batch'-Betrieb, also ohne Benutzerinteraktion, ab. Sie funktionieren nach dem einfachen Prinzip, vom Benutzer definierte Anweisungsfolgen auszulösen, wenn bestimmte Suchmuster im Text (hier die Exportdatei) gefunden werden. Diese Suchmuster können wiederum mit Hilfe von regulären Ausdrücken spezifiziert werden.

```
ZEIT   3566
IP 134.100.15.1 134.100.9.61  136 UDP     0  116
UDP   1020   2049  116  108
```

5.3.4.5 Realisierung der Konfigurierungsfunktionen

Für die Konfigurierung der Transputer (vgl. Abschnitt 4.4.4.3) wurde ein eigenständiges Programm entwickelt. Das Programm realisiert die folgenden Funktionen:

- alle Transputer eines TB in einen definierten Anfangszustand setzen ('Reset' an alle Transputer).
- Programmieren der physikalischen Verbindungstopologie in den Kreuzschienenverteiler auf dem TB (vgl. Abbildung 3.9).

Darüber hinaus gestattet das Programm das

- Anzeigen der zuvor programmierten Verbindungen und
- das Benennen der Transputerlinks mit symbolischen Konstanten für eine leichtere Bedienung.

Die Benennung der Transputerlinks und Buchsen des TB ist der Tabelle 5.2 zu entnehmen. Diese Namen kann der Benutzer zu einem 'Konfigurationsstring' zusammenstellen, der die gewünschten Verbindungen bezeichnet (s.u.). Der Algorithmus bei der Programmierung einer Verbindungstopologie ist in Abbildung 5.17 dargestellt.

Bezeichnung		Symbolischer Name
Transputer 1	Link 0	fest verdrahtet mit 'Hostrechner'
Transputer 1	Link 1	fest verdrahtet mit Kreuzschienenverteiler
Transputer 1	Link 2	A2
Transputer 1	Link 3	A3
Transputer 2	Link 0 – Link 3	B0 – B3
Transputer 3	Link 0 – Link 3	C0 – C3
Transputer 4	Link 0 – Link 3	D0 – D3
Externe Buchse	Nr. 0 – 5	E0 – E5
Konfigurartions-Buchse	Nr. 0 – 5	K0 – K5

Tabelle 5.2: Symbolische Namen für Transputerlinks und Buchsen eines TB

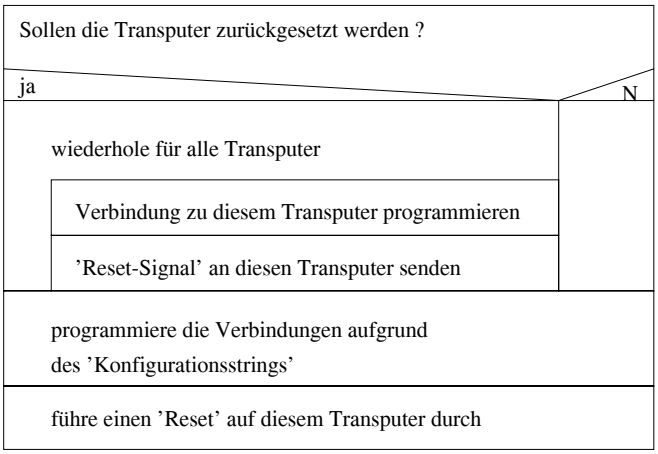


Abbildung 5.17: Algorithmus zum Programmieren einer Verbindungstopologie

Die Angaben, welche Verbindungen zu programmieren sind, kann der Benutzer mit einem 'Konfigurationsstring' an das Programm übergeben. Der folgende Programmaufruf setzt alle Prozessoren in einen definierten Anfangszustand (Parameter -*r*), zeigt die aktuelle Konfiguration an (Parameter -*p*) und programmiert die Topologie aus Abbildung 5.18 (Parameter -*c*).

```
conf -r -p -c A2B0B2D1D2E0A3C0C2D3
```

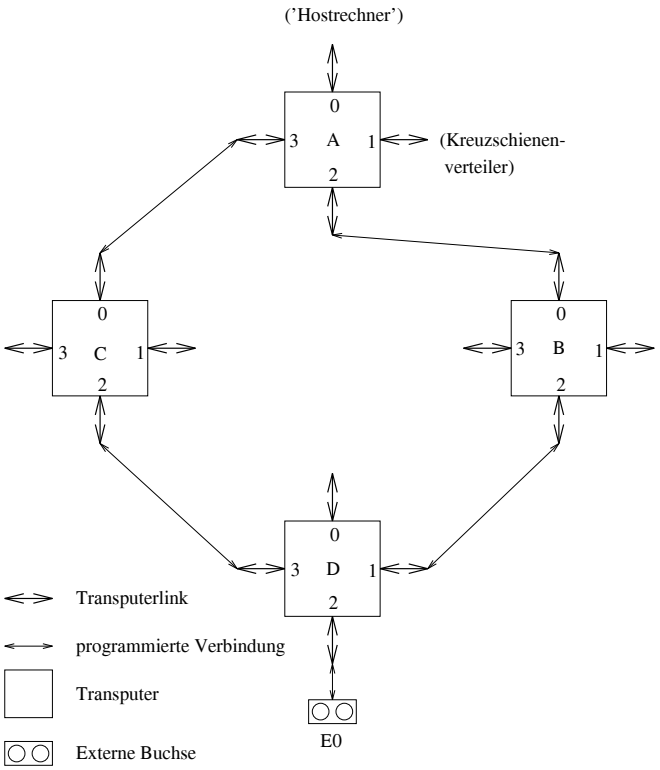


Abbildung 5.18: Topologie "A2B0B2D1D2E0A3C0C2D3"

Nach Ausführung dieses Programms sind alle Transputer in einem definierten Anfangszustand und die vom Benutzer angegebenen Verbindungen sind im Kreuzschienenverteiler programmiert[31].

Neben diesem eigenständigen Programm wurde eine Funktion realisiert, die den Algorithmus aus Abbildung 5.17 für die 6-Transputer-Konfiguration aus Abbildung 3.9 ausführt. Diese Funktion kann über eine Ladefunktion (vgl. Abschnitt 5.3.4.6) von einem Prozeß des 'Hostrechners' gestartet werden.

5.3.4.6 Realisierung einer Ladefunktion

Die Ladefunktion kann von einem Prozeß auf dem 'Hostrechner' benutzt werden, um die Transputer mit einem Programm zu laden.

Der Funktion wird der Dateiname der Programmdatei übergeben sowie eine Nummer zur Identifizierung des 'Transputerboards' (TB). Die Funktion initialisiert den

[31] Wenn man sich die symbolischen Namen aus Tabelle 5.2 als Knoten eines Graphen (G) und die programmierten Verbindungen dazwischen als Kanten des Graphen vorstellt, dann sollte ein Programm, das mit 'iserver' geladen wird, zuvor mit 'config' derart konfiguriert werden, daß die Prozessoren und Verbindungen im 'Konfigurationsskript' einen zusammenhängenden Teilgraphen von G ergeben.

an die PC-Schnittstelle des jeweiligen TB angeschlossenen Transputer in den Anfangszustand und kopiert die Informationen aus der übergebenen Datei über die PC-Schnittstelle auf den Transputer.

Um zu verhindern, daß die Funktion bei Problemen nicht terminiert, startet sie vor dem Laden des Programms einen Zeitgeber, der die Funktion nach Ablauf einer festgelegten Zeit terminiert. Die Funktion meldet den Erfolg oder Mißerfolg an die aufrufende Funktion.

Die Ladefunktion realisiert eine Teilfunktionalität des Programms 'iserver'. Dieses Programm mußte nicht im Rahmen der Diplomarbeit erstellt werden, sondern gehört zu der Entwicklungsumgebung für Transputer (vgl. Abschnitt 3.2.2.2).

Das Programm 'iserver' wird auf dem 'Hostrechner' ausgeführt. Die Hauptaufgabe besteht darin, ein Programm für die Transputer über die PC-Schnittstelle (vgl. Abbildung 3.9) an die Transputer zu übertragen.

Darüber hinaus ermöglicht dieses Programm:

- den an die PC-Schnittstelle angeschlossenen Transputer in einen definierten Anfangszustand zu setzen ('Reset'),
- dem an die PC-Schnittstelle angeschlossenen Transputer Zugriff auf einige Funktionen des 'Hostrechners' (z.B. Ausgaben auf den Bildschirm). Dieser Zugriff auf die Betriebsmittel des Hosts wurden bei der Ladefunktion (s.o.) nicht berücksichtigt.

5.3.4.7 Definieren einer Konfiguration

Auch das Konfigurationsprogramm 'config' gehört zu der Entwicklungsumgebung für Transputer (s.o.). Das Programm ermöglicht es dem Benutzer, wesentliche Teile einer Konfiguration festzulegen. Dazu stellt es folgende Funktionen bereit:

- Benennen von Transputern.
- Benennen von physikalischen Verbindungen zwischen Transputern. Hierbei kann angegeben werden, welche Transputerlinks miteinander verbunden sind. Die Verbindung wird jedoch von diesem Programm nicht hergestellt. Dafür ist die in Abschnitt 5.3.4.5 beschriebene Funktion zu verwenden.
- Benennen von Prozessen und plazieren der Prozesse auf die zuvor benannten Transputer.

Der Benutzer macht diese Angaben in einer Datei (Konfigurationsskript) . Das 'config'-Programm überprüft anhand der Datei, ob die benannten Prozesse (als Binärdateien) existieren und über die Verbindungen auf die entsprechenden Transputer plaziert werden können. Die Prozesse werden dann in eine Programmdatei kopiert und mit einer 'Bootstrap'-Funktion versehen. Diese 'Bootstrap'-Funktion lädt die Prozesse, nach dem Laden der Programmdatei auf den an die PC-Schnittstelle angeschlossenen Transputer (s.o. 'Ladefunktion'), über die physikalischen Verbindungen der Transputer untereinander, auf die vom Benutzer festgelegten Prozessoren.

5.3.5 Realisierung der Filterfunktionen

Bei der Realisierung der Filterfunktionen werden Funktionen zum *Definieren* von Filtern von Funktionen zum *Ausführen* der Filter unterschieden.

5.3.5.1 Realisierung der Definition von Filtern

Die Syntax der Filterdefinition ist durch folgende Grammatik (terminale Symbole sind groß geschrieben, nicht-terminale Symbole sind klein geschrieben) festgelegt:

```
filter -> FILTER simple_filter
simple_filter -> simple_filter LOG_AND simple_filter
simple_filter -> simple_filter LOG_OR simple_filter
simple_filter -> ( simple_filter )
simple_filter -> OFFSET val rel_opr val
simple_filter -> SYMBOL rel_opr SYMBOL
rel_opr ->  < | <= | == | != | >= | >
val -> HEXNUM | DEZNUM
```

Die terminalen Symbole LOG_AND und LOG_OR sollen die entsprechenden logischen Verknüpfungen darstellen. Die unter 'rel_opr' aufgezählten terminalen Symbole stellen die aus der Sprache 'C' bekannten relationalen Operatoren dar. HEZNUM und DEZNUM stehen für skalare Werte, die entweder hexadezimal oder dezimal dargestellt werden können. SYMBOL repräsentiert einen Wert aus einer Symboltabelle.

Der Benutzer tippt die Filterdefinitionen über die Kommandozeile ein. Die Kommandozeile übernimmt die lexikalische und syntaktische Überprüfung der Eingabe (vgl. Abschnitte 4.4.5.2 und 5.3.4.1).

Die Sematik der Regeln soll folgendermaßen sein: Ein skalarer Wert rechts vom 'rel_opr' wird mit den Rohdaten verglichen. Der Vergleich gibt an, ob die durch 'rel_opr' repräsentierte Relation erfüllt ist. Der Vergleich wird bei den Rohdaten ab der Position durchgeführt, die von dem Wert links des 'rel_opr' angegeben wird.

Werden mehrere Vergleiche durchgeführt, so müssen sie durch einen logischen Operator (AND, OR)[32] verknüpft werden.

Das Ergebnis der Auswertung der Filter soll ein boolescher (Wahrheits-)Wert sein, der angibt, ob das Rohdatum der Spezifikation durch den Benutzer genügt.

Die Codegenerierung erzeugt aus den Regeln 'P-Code' für eine Stapelmaschine, die folgende Befehle kennt:

- *'rel_opr'.* Vergleiche gemäß 'rel_opr' und lege das Ergebnis auf den Stapel. Eine Eins repräsentiert einen erfolgreichen Vergleich, eine Null einen nicht erfolgreichen Vergleich.
- *'puschand'.* Verknüpfe die beiden obersten Werte des Stapels mit der logischen Operation 'AND' und lege das Ergebnis wieder auf den Stapel.

[32] Diese Junktorenmenge ist nicht vollständig. Das fehlende (unäre) 'nicht' kann jedoch durch die Klammerung und die relationalen Operatoren ausgedrückt werden, so daß beliebige boolesche Funktionen realisierbar sind.

- *'pushor'.* Verknüpfe die beiden obersten Werte des Stapels mit der logischen Operation 'OR' und lege das Ergebnis wieder auf den Stapel.
- *'pop'.* Hole den obersten Wert vom Stapel.

Dazu wird während der syntaktischen Überprüfung der Benutzereingabe ein Ableitungsbaum erstellt, der es einer anschließenden Funktion für die 'P-Code-Generierung' ermöglicht, die Infix-Ausdrücke in die für die Stapelmaschine notwendige Postfix-Notation umzuwandeln.

5.3.5.2 Realisierung eines 'P-Code-Interpreters' für Filterausdrücke

Anhand eines Beispiels wird das Ergebnis der Codegenerierung verdeutlicht. Der folgende Ausdruck

```
filter offset 0x12 == 0x0815 and
     ( offset 0x20 < 0x64 or offset 0x22 >= 0x20 )
```

mit der natürlich-sprachlichen Semantik: *" Vergleiche, ob ab dem Offset 0x20 ein Wert kleiner 0x64 steht oder ob ab dem Offset 0x22 ein Wert größer oder gleich 0x20 steht. Mindestens eine dieser beiden Bedingungen muß erfüllt sein. Zusätzlich vergleiche, ob ab dem Offset 0x12 der Wert 0x0815 steht."* wird in das "Programm" aus Abbildung 5.19 übersetzt. Dieses Programm muß in einer Nachricht vom 'Hostrechner' an den Prozeß 'PDEV' gesendet werden, der den 'P-Code' ausführt (s.u.).

'>='	0x22	0x20
'<='	0x20	0x64
'pushor'		
'=='	0x12	0x0815
'pushand'		
'pop'		

Abbildung 5.19: 'P-Code' eines Filterausdrucks

Die Vergleiche wurden durch entsprechende Funktionsaufrufe realisiert. Die ebenfalls auf Funktionsaufrufe abgebildeten Operationen 'pushand', 'pushor' und 'pop' arbeiten auf einem ('last in first out') Stapel. Der eigentliche 'P-Code'-Interpreter arbeitet die Befehle nacheinander ab und setzt sie in Funktionsaufrufe um.

Bei den einzelnen Vergleichen wird entsprechend der Kardinalität des zweiten Argumentes verglichen. In obigem Beispiel wird demnach zweimal ein Byte verglichen (0x20 und 0x64) und einmal zwei Byte (0x0815).

5.4 Realisierung der unterstützenden Funktionen

Auf die Realisierung der Vermittlungs- und Verteilungsfunktionen wird in Abschnitt 5.4.1 eingegangen. Die Kommunikationsunterstützung für den 'Hostrechner' wird in Abschnitt 5.4.4 beschrieben. Anmerkungen zu weiteren unterstützenden Funktionen folgen in Abschnitt 5.5.

5.4.1 Realisierung der Vermittlungsfunktionen

Die Vermittlungsfunktionen sollen den Nachrichtenaustausch zwischen den Transputern (und dem 'Hostrechner') vereinfachen. Die Struktur der Nachrichten muß einem festgelegten Nachrichtenformat genügen, um von den Vermittlungsfunktionen richtig interpretiert zu werden.

5.4.1.1 Nachrichtenformat

Um zu verstehen, wie Nachrichten vermittelt werden, muß zuvor das Nachrichtenformat für die Interprozessorkommunikation vorgestellt werden. Alle Nachrichten beginnen mit einer Längenangabe, die die Gesamtlänge der Nachricht angibt. Der Längenangabe folgt eine Markierung (engl. 'Tag'), die den Typ der Nachricht angibt und gleichzeitig den oder die Empfangsprozesse adressiert. Der Nachrichtentyp wird vom Benutzer durch ein 16-Bit-Wort angegeben. Er hat jedoch keine Auswirkung auf das Vermitteln der Nachricht. Die verbleibenden 16 Bit des 'Tags' entsprechen jeweils einer Prozeßgruppe oder einer 'Richtung'. Sie können einzeln gesetzt und somit frei kombiniert werden. Daraus ergeben sich 65536 verschiedene Nachrichtentypen und 16 verschiedene Prozeßgruppen bzw. Richtungsangaben für die Vermittlungsfunktion. Das allgemeine Nachrichtenformat ist in Abbildung 5.20 dargestellt.

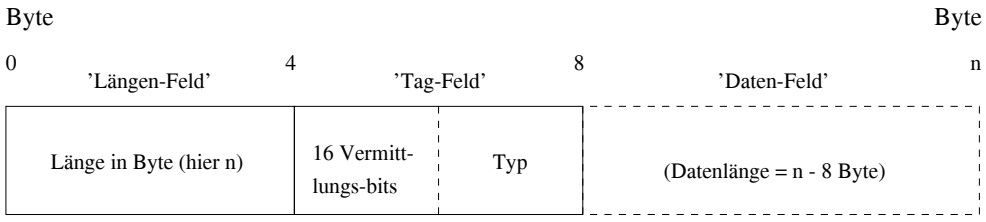


Abbildung 5.20: Allgemeines Nachrichtenformat

5.4.1.2 Vermittlungsalgorithmus

Jeder Prozessor verfügt über eine eigene Vermittlungstabelle. Darin sind folgende Angaben enthalten:

- Anzahl der zu unterstützenden Kommunikationskanäle,

- Bezeichnung dieser Kommunikationskanäle,
- Angaben, welche Prozeßgruppen bzw. Richtungen über den jeweiligen Kommunikationskanal erreichbar sind.

Eine eintreffende Nachricht wird nun über alle Kommunikationskanäle ausgegeben, bei deren Tabelleneintrag mindestens eine Prozeßgruppe bzw. Richtungsangabe mit der im 'Nachrichten-Tag' übereinstimmt. Dazu ist nur eine 'AND'-Verknüpfung von Tabelleneintrag und 'Nachrichten-Tag' erforderlich. Die Vermittlungsentscheidung kann sehr schnell getroffen werden. Die Nachricht wird jedoch nie über denselben Kanal ausgegeben, über den sie eingetroffen ist. Dadurch werden endlos laufende, sich ständig vervielfachende Nachrichten vermieden.

Durch diesen einfachen Algorithmus lassen sich die Nachteile einer 'ad hoc' Lösung vermeiden (vgl. Abschnitt 4.5.1). Durch getrennte Prozesse für das Senden und Empfangen von Nachrichten vermeidet man außerdem Verklemmungen und trennt die Kommunikation von anderen Berechnungen auf dem Prozessor (vgl. [Zit91a, S. 99], [Ben94, S. 114f]). In Abbildung 5.21 ist der Algorithmus zum Vermitteln von Nachrichten dargestellt.

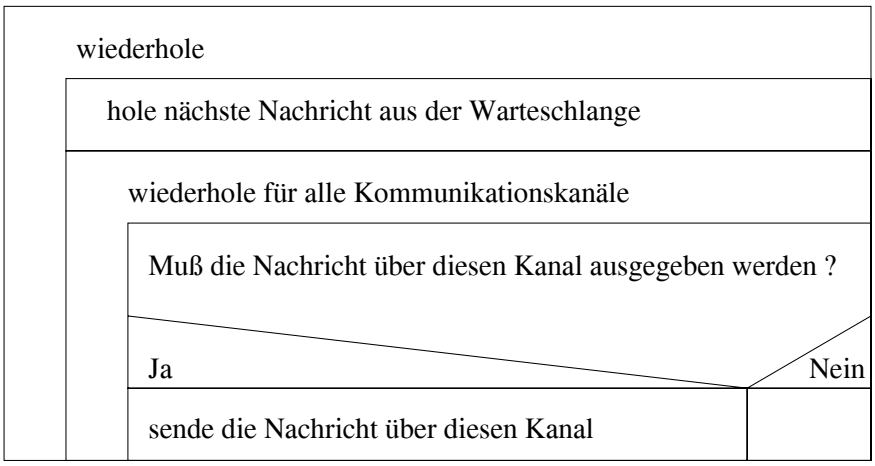

wiederhole
hole nächste Nachricht aus der Warteschlange
wiederhole für alle Kommunikationskanäle
Muß die Nachricht über diesen Kanal ausgegeben werden ?
Ja
Nein
sende die Nachricht über diesen Kanal

Abbildung 5.21: Vermitteln von Nachrichten

Ein Prozeß, der eine Nachricht versenden möchte, spezifiziert mit den 'Vermittlungs-bits' welche Prozesse seine Nachricht empfangen sollen. Danach übergibt er die Nachricht der Sendefunktion, die die Weitervermittlung übernimmt. Nachrichten anderer Prozesse, die aufgrund ihrer 'Vermittlungs-bits' nicht für die lokalen Prozesse eines Prozessors bestimmt sind, werden für die lokalen Verarbeitungsprozesse völlig transparent (ohne deren Mithilfe) weitervermittelt.

Die Vorteile dieser allgemeinen Vermittlungsfunktionen lassen sich wie folgt zusammenfassen:

- Vermeiden von Verklemmungen beim Nachrichtenaustausch,

- Vermitteln von Nachrichten an Prozeßgruppen über beliebig viele Prozessoren,
- nebenläufiges Senden, Empfangen und Verarbeiten der Nachrichten,
- flexible Anpaßbarkeit an neue Konfigurationen durch Vermittlungstabellen,
- relative Adressierung einzelner Prozeßgruppen durch 'Richtungsangaben' möglich (z.B. "Nachricht an alle Prozesse 'über' mir"),
- Lokationstransparenz: Ein Prozeß kann seine Nachricht unabhängig davon vermitteln lassen, ob der Empfangsprozeß auf demselben Prozessor plaziert wurde oder auf einem anderen.

Der Prototyp benutzt sieben der 16 'Vermittlungs-bits' (VBs). Sollte sich bei Erweiterungen herausstellen, daß 16 VBs nicht ausreichen, um alle Prozeßgruppen zu adressieren, gibt es zwei Möglichkeiten für eine Erweiterung des Adreßbereiches:

- Das 'Tag-Feld' (vgl. Abbildung 5.20) kann beispielsweise auf 64 bit vergrößert werden. Man hätte dann 48 VBs. Diese Methode würde nur kleine Änderungen in den Programmquellen erfordern.
- Indizieren einer Vermittlungstabelle mit den vorhandenen VBs. Die VBs würden als Zahl interpretiert werden, die als Index in eine Vermittlungstabelle mit Wegwahlinformationen dient. Mit den vorhandenen 16 VBs könnte man maximal 65536 Einträge in der Vermittlungstabelle indizieren[33].

Durch die oben getroffene Wahl, jedem VB eine Prozeßgruppe zuzuordnen, ist die Vermittlungstabelle sehr einfach strukturiert und überschaubar.

5.4.2 Realisierung der Verteilungsfunktionen

Bei diesem Prototypen sollen eintreffende Dateneinheiten gleichmäßig[34] auf die Paketträger aufgeteilt werden. Außerdem sollen die Dateneinheiten abwechselnd[35] auf beide Paketträger verteilt werden.

Auf Grundlage der Vermittlungsfunktionen ist die Verteilung trivial realisierbar. Es wird ein 'Vermittlungs-bit' für die Prozeßgruppe der Paketträger mit einem weiteren 'Vermittlungs-bit' kombiniert, das die jeweilige 'Richtung' des Empfängers (einer der beiden Paketträger) bezüglich des Senders angibt. Bei jeder neuen Dateneinheit wird die Richtung geändert, so daß sich die oben gewählte Aufteilung ergibt.

Auch die Verteilungsfunktionen lassen sich —für den allgemeinen Fall mit mehreren Paketträgern ($>> 2$)— hierarchisch anordnen. Bei dem eben dargestellten Verteilungsverfahren würde man einen binären Baum erhalten, an dessen inneren Knoten

[33] Diese Anzahl gilt pro Prozeß, da jeder Prozeß über eine eigene Vermittlungstabelle verfügen kann. Es wären sehr große Analysewerkzeuge realisierbar!

[34] Das muß nicht zwangsläufig so sein. Eine andere Verteilung könnte ebensogut lastabhängig oder zufällig erfolgen.

[35] Auch hier sind andere Vorgehensweisen denkbar. Dateneinheiten könnten gesammelt und dann schlagartig (in einem 'burst') an einen Paketträger weitergegeben werden.

Verteilungsfunktionen die Dateneinheiten auf die 'Blätter' (hier die Paketträger) verteilen (vgl. Abbildung 5.22).

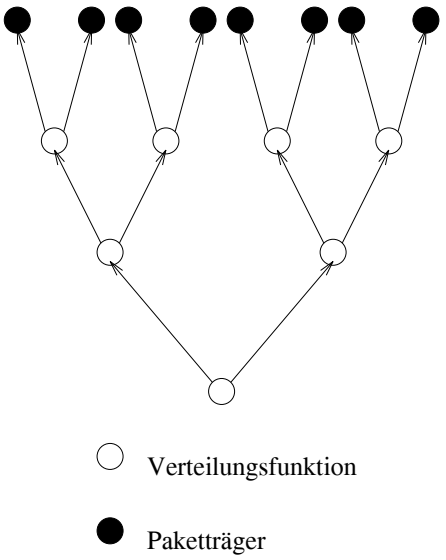


Abbildung 5.22: Verteilen von Dateneinheiten auf mehrere Paketträger

5.4.3 Anmerkung zu Verbindungsmanagement-Funktionen

Die Verbindungsmanagement-Funktionen wurden in Abschnitt 4.5.2 entworfen. In Abschnitt 5.3.2.7 wurde dargestellt, wie derartige Funktionen für das TCP-Protokoll realisiert werden könnten[36]. Diese Funktionen konnten leider nicht implementiert werden, da die verbleibende Zeit (bis zur Abgabe der Diplomarbeit) für die Bewertung des Prototypen benötigt wurde (vgl. Kapitel 7).

5.4.4 Realisierung der Kommunikationsunterstützung für den 'Hostrechner'

Beim Entwurf der Kommunikationsunterstützung wurde bereits die Entscheidung getroffen, keinen vollständigen Gerätetreiber für OS/2 zu implementieren. Als Grund wurde der große Aufwand für die Programmierung eines Gerätetreibers genannt (vgl. Abschnitt 4.5.3). Ein weiterer Grund ist, daß nur veraltete Dokumentation zur Verfügung steht (vgl. [Int89]). Die Dokumentation beschreibt eine alte 16-bit-Version von OS/2 und ist für die Programmierung unter OS/2 Version 3 nicht zu gebrauchen.

Die Nachteile der Realisierung eines vollständigen Gerätetreibers konnten umgangen werden, indem eine Besonderheit des Betriebssystems ausgenutzt wurde. Das

[36] 'TCP' ist das einzige verbindungsorientierte Protokoll, das vom Prototypen analysiert werden kann.

Betriebssystem gestattet eine Umgehung der Kern-Funktionen beim Zugriff auf Ein- und Ausgabeadressen. Anwendungsprogramme können für bestimmte Adreßbereiche Zugriffsprivilegien erlangen, um so unter Umgehung der Kern-Funktionen direkt auf Ein/- Ausgabeadressen bestimmter Geräte zuzugreifen. Darüber hinaus kann man eigene Aufrufkonventionen beibehalten. Außerdem ist kein Prozeßwechsel erforderlich, da keine Kern-Funktionen benutzt werden.

Diese Funktionen wurden, wie oben dargestellt, unter Umgehung der Kern-Funktionen realisiert. Dazu war es nötig, den Programmcode der Funktionen in eigenständige privilegierte 'Codesegmente' zu übersetzen[37]. Die Sende- und Empfangsbereitschaft des ersten Transputers kann nun durch Zugriff auf die dafür vorgesehene Ein/- Ausgabeadressen überprüft werden. Abbildung 5.23 zeigt diesen vereinfachten Zugriff auf Geräte. Man erkennt die Umgehung der Kern-Funktionen (gestrichelt umrandet) durch Programmcode in privilegierten 'Segmenten'.

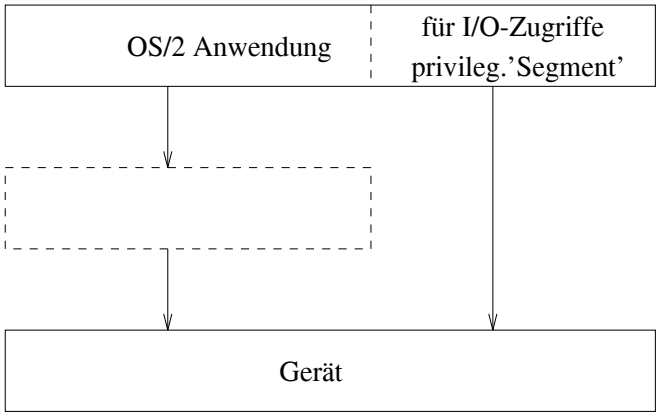


Abbildung 5.23: Vereinfachter Zugriff auf Geräte

Bei der Realisierung der Funktionen zum Senden und Empfangen von Nachrichten, mußte die Sende- bzw. Empfangsbereitschaft für jedes zu übertragene Byte überprüft werden, da die Zugriffsgeschwindigkeit des 'Hostrechners' auf die Ein/- Ausgabeadressen größer ist als die Geschwindigkeit bei der Übertragung der Byte von und zum Transputer.

Die Sende- und Empfangsfunktionen wurden so realisiert, daß sie Nachrichten verarbeiten können, die dem Format aus Abbildung 5.20 entsprechen. Damit ist sichergestellt, daß die Interprozeßkommunikation unabhängig davon ist, ob Quelle oder Senke der Nachricht ein Prozeß auf dem Transputer oder auf dem 'Hostrechner' ist. Als Beispiel ist in Abbildung 5.24 das Empfangen einer Nachricht vom Transputer auf dem 'Hostrechner' dargestellt.

[37] Dadurch ist es möglich, Funktionen dieses 'Codesegments' in einem privilegierten Modus auszuführen, so daß Zugriffe auf die (normalerweise vom Betriebssystem) geschützten Ein/- Ausgabeadressen keine Schutzverletzungen darstellen.

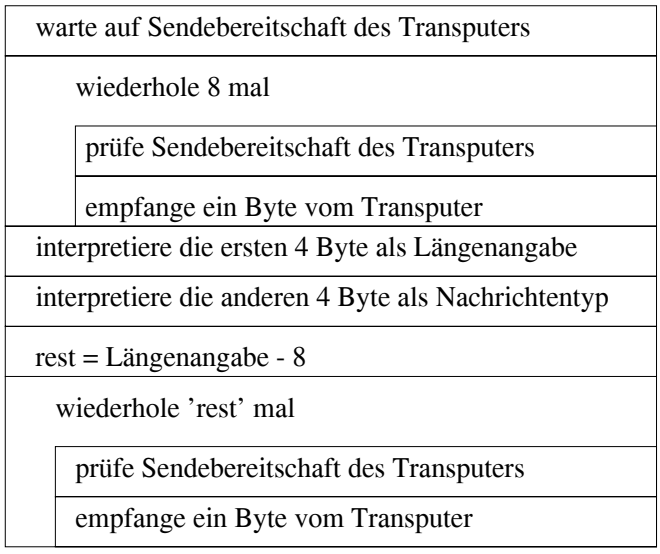

warte auf Sendebereitschaft des Transputers
wiederhole 8 mal
prüfe Sendebereitschaft des Transputers
empfange ein Byte vom Transputer
interpretiere die ersten 4 Byte als Längenangabe
interpretiere die anderen 4 Byte als Nachrichtentyp
rest = Längenangabe - 8
wiederhole 'rest' mal
prüfe Sendebereitschaft des Transputers
empfange ein Byte vom Transputer

Abbildung 5.24: Empfangen einer Nachricht auf dem 'Hostrechner'

5.5 Anmerkungen zu weiteren Funktionen

In Abschnitt 4.6 wurden weitere unterstützende Funktionen entworfen. In den folgenden Abschnitten wird dargestellt, wie diese Funktionen zur Unterstützung anderer Funktionen eingesetzt werden.

5.5.1 Realisierung der Ringpuffer-Verwaltung

Sämtliche Analyseergebnisse und sonstige Nachrichten, die von den Transputern an den 'Hostrechner' gesendet werden, legen die Kommunikationsfunktionen (vgl. Abschnitt 5.4.4) in einem Ringpuffer ab.

Die in Abschnitt 5.4.4 beschriebene Empfangsfunktion (vgl. auch Abbildung 5.24) legt empfangene Nachrichten in den Ringpuffer. Um anderen Funktionen einen einfachen Zugriff auf die Daten im Ringpuffer zu ermöglichen, werden die Nachrichten jeweils zusammenhängend abgelegt. Dadurch ist sichergestellt, daß Nachrichten der Transputer nicht am Ringpufferende zerteilt werden. Funktionen, die auf diese Nachrichten zugreifen wollen, brauchen keine Überprüfungen der 'Lage' der Nachrichten im Ringpuffer vorzunehmen.

Da das Empfangen von Nachrichten möglicherweise durch einen anderen Prozeß realisiert wird, muß der Zugriff auf den Ringpuffer durch eine Semaphore geschützt werden. Durch diese Semaphore wird sichergestellt, daß Nachrichten erst vollständig empfangen werden, bevor sie von anderen Funktionen verarbeitet werden.

Der gesamte Zugriff auf den Ringpuffer wird durch folgende Funktionen realisiert:

- Ringpuffer initialisieren. Diese Funktion reserviert den Speicher für den Ring-

puffer, initialisiert die Semaphore für die Zugriffssynchronisation und initialisiert globale Variablen.

- Test auf vorhandene Nachrichten. Diese Funktion zeigt an, ob es unverarbeitete Nachrichten im Ringpuffer gibt.
- Test auf freien Speicherplatz. Diese Funktion berechnet den freien Speicherplatz im Ringpuffer, gibt also die maximale Größe einer zu speichernden Nachricht zurück.
- Position der nächsten Nachricht bestimmen. Diese Funktion liefert einen Zeiger auf die älteste unverarbeitete Nachricht im Ringpuffer.
- Nachrichten kopieren und löschen.
- Nachrichten in den Ringpuffer einschreiben. Diese Funktion wird durch die Empfangsfunktion aus Abschnitt 5.4.4 realisiert.

5.5.2 Realisierung der Symboltabellen-Verwaltung

Der Benutzer hat die Möglichkeit, über eine Datei Symbole zu definieren. Diese Symbole kann er beispielsweise bei Eingaben über die Kommandozeile verwenden. Der Zugriff auf die Werte und Attribute der Symbole wird über eine Streufunktion[38] realisiert. Falls zwei oder mehr Symbole durch die 'Hash-Funktion' auf denselben Wert abgebildet werden, werden die Symbole mit ihren Attributen in einer Liste angeordnet.

Diese Funktion wird wiederum in Funktionen benutzt, die

- Symbole und deren Attribute in einer Symboltabelle ablegen,
- Symbole und deren Attribute aus der Symboltabelle zurückgewinnen,
- die Existenz von Symbolen überprüfen, um unbeabsichtigtes Neubelegen vorhandener Attribute zu vermeiden.

5.6 Vorgehen bei Erweiterungen

In diesem Abschnitt werden einige Hinweise gegeben, wie der Prototyp erweitert werden könnte. Die allgemeine Vorgehensweise wird in Abschnitt 5.6.1 diskutiert. Ein konkretes Beispiel folgt in Abschnitt 5.6.2.

5.6.1 Erweitern des Prototypen um neue Transputer-Prozesse

Grundlage dieser Erweiterungen sollte das 'Prozeß-Template' aus Abbildung 5.3 sein. Es werden in jedem Fall die Vermittlungsfunktionen und eine Ablaufkoordinierung benötigt. Die weiteren Schritte sind in der folgenden Liste zusammengefaßt:

[38] Diese 'Hash-Funktion' sichert auch bei einer großen Anzahl von Symbolen einen schnellen Zugriff auf die Attribute.

- Festlegen der Prozeßklasse[39] für die Gruppenadressierung. Hierbei kann es sich um eine neue Klasse handeln, um eine Kombination von bereits vorhandenen Klassen oder um eine neue Klasse, die zusätzlich mit vorhandenen kombiniert wird.
- Erstellen einer oder mehrerer Einträge in die Vermittlungstabelle für Prozesse dieser neuen Klasse. Einerseits müssen alle Nachrichten an die eigene Klasse empfangen werden können, andererseits müssen alle Nachrichten an andere Prozeßklassen, die möglicherweise empfangen werden können, transparent weitervermittelt werden.
- Festlegen der zu verarbeitenden Nachrichten. Die zu verarbeitenden Nachrichten bestimmen wesentlich die Komplexität der Ablaufkoordinierung dieser neuen Prozeßklasse. In jedem Fall sollte auf die Anforderung von Statusinformationen reagiert werden können.
- Implementieren des neuen Prozesses. Für jede zu verarbeitende Nachricht ist eine entsprechende Funktion zu realisieren.
- Anpassen der Konfiguration über ein entsprechendes Skript.

Diese Arbeitsschritte fallen immer an, wenn eine neue Klasse von Prozessen realisiert wird. Zusätzlich müssen noch Änderungen an bestehenden Prozeßklassen vorgenommen werden, wenn Prozesse dieser Klasse mit Prozessen der neuen Klasse kooperieren sollen. Eventuell müssen auch vorhandene Einträge in der Vermittlungstabelle geändert werden, um die neue Prozeßklasse in eine geplante Konfiguration zu integrieren.

Erweiterungen an vorhandenen Prozeßklassen sind in der Regel deutlich aufwandsärmer, da meist nur neue Nachrichten definiert werden, für die dann entsprechende Funktionen der Ablaufkoordinierung bereitzustellen sind.

5.6.2 Erweitern des Werkzeuges um die Prozeßklasse 'PNOTHING'

Um die Arbeitsschritte zu verdeutlichen, wird eine Erweiterung um die imaginäre Prozeßklasse 'PNOTHING' skizziert. Ein Prozeß dieser Klasse soll nur auf Statusmeldungen antworten können (Minimalanforderung).

- Die Prozeßklasse ist 'PNOTHING'. Dieser Klasse wird ein bisher unbenutztes Vermittlungs-bit zugeordnet. Später ist durch setzen dieses Bits in Nachrichten eine Gruppenadressierung der neuen Klasse möglich (vgl. Abschnitt 5.4.1.1).
- Es soll nur einen Prozeß dieser Klasse geben, der zwischen 'PSTAT' und 'Hostrechner'-Prozeß plaziert werden soll. Es muß daher einen Eintrag in die Vermittlungstabelle für diesen Prozeß geben, der alle Nachrichten des 'Hostrechners' nach 'PSTAT' und alle Nachrichten von 'PSTAT' zum 'Hostrechner' weiterleitet.

[39] Es gibt zur Zeit vier Prozeßklassen. Alle oben dargestellten Prozesse gehören einer eigenen dedizierten Klasse an. Dazu kommt noch die Prozeßklasse für den 'Hostrechner'-Prozeß.

Dazu übernimmt man einerseits die entsprechenden Kombinationen von 'PSTAT' und setzt andererseits das Vermittlungs-bit für Nachrichten an den 'Hostrechner'. Der neue Prozeß verhält sich damit transparent gegenüber Nachrichten anderer Prozesse untereinander. Für den internen Kanal werden die Vermittlungs-bits für 'Broadcast'-Nachrichten und 'Multicast'-Nachrichten an die eigene Klasse gesetzt (das Bit, das oben 'PNOTHING' zugeordnet wurde).

- Es soll nur auf Statusanforderungen vom 'Hostrechner' geantwortet werden.
- Es wird eine Funktion realisiert, die eine Volltextnachricht an den 'Hostrechner'-Prozeß[40] generiert. Der Text soll lauten: "PNOTHING OK". Die Nachricht wird als Antwort auf die Statusanforderung des 'Hostrechners' generiert.
- Das vorhandene Konfigurationsskript (vgl. Abschnitt 5.3.4.5) wird so abgeändert, daß eine Instanz der neuen Prozeßklasse 'zwischen' 'Hostrechner'-Prozeß und 'PSTAT' plaziert wird[41].

Falls vom Benutzer eine Statusinformation angefordert wird, sollten sich die anderen Prozesse mit ihren Statusinformationen melden. Zusätzlich sollte die Meldung "PNOTHING OK" auf dem Terminal ausgegeben werden.

Erweiterungen um zusätzliche Prozesse einer vorhandenen Klasse beschränken sich auf Ergänzungen an der Vermittlungstabelle und Schreiben neuer Konfigurationsskripte.

5.7 Anmerkungen zur Realisierung und zu Erweiterungen

Um künftige Erweiterungen des Prototypen vorzubereiten, wurden die Programmquellen modular programmiert und dokumentiert. Das Kapitel 5 ist als Teil der Dokumentation zu verstehen. Angaben zu besonderen Algorithmen und die Beschreibung der verwendeten Datenstrukturen und abstrakten Datentypen sind in den Programmquellen enthalten. Die Verwaltung der Programmquellen mit einem Versionskontrollsystem wird in Abschnitt 5.7.1 beschrieben. In Abschnitt 5.7.2 werden einige Erweiterungsvorschläge dargestellt. Abschnitt 5.7.3 zeigt zwei Möglichkeiten zur Integration des Prototypen in 'Managementsysteme'. Eine Abgrenzung des Prototypen von anderen Werkzeugen der Kommunikationsmeßtechnik folgt in Abschnitt 5.7.4.

5.7.1 Verwaltung der Programmquellen mit einem Versionskontrollsystem

Die einzelnen Programmodule sind natürlich nicht sofort in ihrer vorläufigen Endfassung[42] entstanden, sondern durch sukzessive Erweiterungen und Fehlerbehebungen.

[40] Der Nachrichtentyp ist 'TEXTINFO', das zu setzende Vermittlungs-Bit ist 'ROUTE_USER'.
[41] Dazu ist eine Erweiterung um einen Transputer erforderlich!
[42] Das ist die Fassung, in der der Prototyp zur Zeit (Februar 1996) vorliegt.

Die einzelnen Erweiterungen und Fehlerbehebungen an den Modulen sind ebenfalls dokumentiert. Dazu wurde ein Versionskontrollsystem (vgl. [Tic85]) verwendet, das neben der Kommentierung einer Änderung für eine kompakte Speicherung der Quellen sorgt.

Ein weiterer Vorteil durch die Verwendung der Versionskontrolle ist, daß zu einer beliebigen Version zurückgekehrt werden kann. Dadurch sind nicht nur Erweiterungen möglich, sondern auch alternative Realisierungen bereits vorhandener Funktionen auf Grundlage einer beliebigen Version. Bei Gruppenarbeiten verhindert das Versionskontrollsystem (VKS) darüber hinaus, daß mehrere Programmierer gleichzeitig Änderungen an demselben Modul vornehmen.

Der folgende Text ist eine Auszug der Kommentare zu einem Modul (Ablaufsteuerung des Prozesses 'PPAKET''). Das VKS trägt zu jeder Version das Erstellungsdatum und die Uhrzeit sowie den Namen des Programmierers ein, der die Änderung an den Programmquellen vorgenommen hat. Zusätzlich kann der Programmierer einen Kommentar zu jeder Version eingeben, der die Änderungen an dem jeweiligen Modul dokumentieren sollte.

```
RCS file: RCS/ppaket.c,v
Working file: ppaket.c
head: 2.7
branch:
locks: strict
benecke: 2.7
access list:
symbolic names:
keyword substitution: kv
total revisions: 21; selected revisions: 21
description:
Erstellt von Carsten Benecke im Rahmen seiner Diplomarbeit.
----------------------------
revision 2.7 locked by: benecke;
date: 1996/01/26 15:46:21;  author: benecke;  state: Exp;  lines: +66 -6
Messunterstuetzung fuer interne Auswertung
C.B.
----------------------------
revision 2.5
date: 1995/12/08 13:58:45;  author: benecke;  state: Exp;  lines: +9 -3
Aenderungen an RESET-Bearbeitung.
----------------------------
revision 2.4
date: 1995/11/22 09:19:17;  author: benecke;  state: Exp;  lines: +132 -99
Endversion 3tp, 4tp.
----------------------------
revision 2.3
date: 1995/11/20 12:44:13;  author: benecke;  state: Exp;  lines: +23 -8
Erste lauffaehige Version ohne dynamischen Nachrichtenspeicher.
C.B.
----------------------------
revision 2.0
date: 1995/11/16 10:46:26;  author: benecke;  state: Exp;  lines: +255 -272
branches:  2.0.1;
Schnelle Ringpuffer anstatt dynamischen Speicher fuer Nachrichten.
```

```
C.B.
----------------------------
revision 1.10
date: 1995/11/06 13:11:39;  author: benecke;  state: Exp;  lines: +29 -20
Benutzerauswahl (2. Filter) wird beachtet.
----------------------------
revision 1.9
date: 1995/11/06 11:09:44;  author: benecke;  state: Exp;  lines: +14 -2
Vor Einbau des Filters 2. Stufe
----------------------------
revision 1.8
date: 1995/10/26 15:59:56;  author: benecke;  state: Exp;  lines: +376 -376
Erweitert um Analysefunktion fuer ICMP - Nachrichten.
C.B.
----------------------------
revision 1.7
date: 1995/09/27 21:33:47;  author: benecke;  state: Exp;  lines: +73 -9
Erweiterungen fuer IP.
----------------------------
revision 1.5
date: 1995/09/15 19:15:15;  author: benecke;  state: Exp;  lines: +104 -5
ARP-Protokollanalyse eingebaut (ARP/RARP)
C.B.
----------------------------
revision 1.4
date: 1995/09/13 19:11:32;  author: benecke;  state: Exp;  lines: +17 -9
Fehler in Funktion eth_analyse behoben.
C.B.
----------------------------
revision 1.3
date: 1995/09/11 16:12:33;  author: benecke;  state: Exp;  lines: +9 -5
ac_sym.c bei vorheriger Version (1.2) und dieser (1.3) unveraendert.
message.c und message.h sind jedoch veraendert worden.
Diese Version behebt einen Fehler in der Interpretation des
Ethernet-Typ-Feldes. (short int) der Transputer != (short int) der
'Network-Byte-Order'.
C.B. 11.09.95
----------------------------
revision 1.2
date: 1995/09/11 14:13:18;  author: benecke;  state: Exp;  lines: +50 -2
Einfache Analysefunktionen fuer Ethernet eingebaut. Neue
Versionen von mtypes.h net.h net.c und ac_sym.c erforderlich.
C.B. 11.09.95
----------------------------
revision 1.1
date: 1995/09/08 17:51:28;  author: benecke;  state: Exp;
Initial revision
----------------------------
revision 2.0.1.1
date: 1995/11/16 15:49:21;  author: benecke;  state: Exp;  lines: +20 -8
muesste laufen, tut es aber nicht
=============================================================================
```

5.7.2 Wichtige nicht realisierte Funktionen

Die Anforderungen an den Prototypen aus Abschnitt 2.7.4 sind erfüllt worden. Ziel dieser Anforderungen war es, einen Prototypen zu spezifizieren, der altbekannte Aufgaben aus der Kommunikationsmeßtechnik auf Grundlage von Parallelverarbeitung erfüllt. Bei der Realisierung wurde besonders viel Wert auf die Implementierung der Unterstützungsfunktionen für das Konzept der Paketparallelität gelegt.

Dieses Vorgehen ist bei der Realisierung eines Prototypen verständlich. Es müssen primär die grundlegenden Funktionen realisiert werden, bevor praxisnahen Anforderungen nachgegangen werden kann. Die Erweiterbarkeit und die Skalierbarkeit des Prototypen sind auf Grundlage dieser Gewichtung gewährleistet.

Andererseits muß man anmerken, daß die geringe Anzahl analysierbarer Protokolle und die sehr eingeschränkten Analysefunktionen kaum praxisnahen Anforderungen gerecht werden. Hier ist viel Raum für Erweiterungen. Die wichtigsten bisher nicht realisierten Funktionen sind:

- Hilfsfunktionen. Das Werkzeug sollte idealerweise 'selbsterklärend' sein. Ein wichtiger Schritt in diese Richtung ist die Implementierung einer kontextsensitiven 'on line'-Hilfsfunktion. Unter OS/2 ist dies verhälnismäßig leicht, da die Realisierung von Hilfsfunktionen von den Entwicklungswerkzeugen unterstützt wird.

- Analysefunktionen für anwendungsnahe Protokolle und Multi-Media-Protokolle. Es sind bisher nur Analysefunktionen für netznahe Protokolle (Ethernet, IEEE 802.3, ARP, RARP) und Protokolle auf der Vermittlungs- und Transportschicht (IP, UDP, ICMP, TCP) realisiert worden. Ein wesentlicher Vorteil neben der prinzipiellen Erweiterbarkeit ist, daß parallel zur Realisierung eines neuen Protokolls eine Analysefunktion für dieses Protokoll realisiert werden kann. Ist der Prototyp des Protokolls einsatzbereit, kann sofort mit Messungen begonnen werden.

- Funktionen für die Programmierung von 'Trigger' (vgl. Abschnitt 2.2). Die Möglichkeit benutzerspezifizierte Aktionen in Abhängigkeit von bestimmten Ereignissen bei der Messung auszulösen, wäre wünschenswert für Netzmanagement-Aufgaben.

- 'Slicing'-Techniken. Beim 'Slicing' werden die Dateneinheiten nicht vollständig an die Analyse weitergegeben, sondern nur eine vom Benutzer festzulegende Anzahl Byte[43]. Im Kontext von HSN läßt sich die Last durch Rohdaten im Werkzeug durch 'Slicing' deutlich reduzieren.

Werden die oben genannten Funktionen realisiert, wäre das Werkzeug durchaus praxistauglich. Die Funktionen müßten in Zusammenarbeit mit den potentiellen Benutzern

[43] Beispielsweise könnte der Benutzer die Analyse von UDP-Dateneinheiten verlangen. Bei einem Ethernet-basierten Werkzeug würden durch 'Slicing' nur die ersten 50 Byte eines Ethernet-Frames an die Analyse weitergegeben werden. Das entspricht 22 Byte Ethernet-Kopf + 20 Byte IP-Kopf + 8 Byte UDP-Kopf.

entwickelt werden.

5.7.3 Erweiterungsvorschläge

Primär sollten die im letzten Abschnitt (5.7.2) genannten Funktionen realisiert werden. Darüber hinaus bietet sich die Integration des Meßwerkzeuges in ein Netzwerk-Managementsystem (NMS) an. Durch eine Integration des Meßwerkzeugen in ein NMS könnte man die Messungen automatisieren und unter die Kontrolle des NMS stellen. Vorschläge für die Architektur eines derartigen NMS sind beispielsweise in [MSS93].

Eine Möglichkeit diese Integration zu erreichen, wäre die Übertragung von Kommandos an das Meßwerkzeug über das Netz. Durch adressieren des Meßwerkzeuges[44] könnte die Interpretation der Dateneinheit als 'Kommando(s)' veranlaßt werden. Das Meßwerkzeug könnte so als 'intelligente' Meßstation[45] in einem Verteilten System eingesetzt werden.

Die erfaßten Rohdaten und die Analyseergebnisse könnten unter Verwendung des 'Simple Network Management Protocol' (SNMP) vom Meßwerkzeug in eine (verteilte) Datenbank abgelegt werden. Diese Integrationsmethode wird beispielsweise in [Par95] diskutiert. Eine Anwendung eines solchen NMS ist die Überwachung von zugesicherter Dienstgüte, auch 'Quality of Service' (QoS) genannt (vgl. z.B.[FY94]).

Eine weitere Integrationsmöglichkeit für Meßwerkzeuge wird in [D+94] diskutiert. Das Meßwerkzeug dient hier der modellgestützten Bewertung von Verteilten Systemen. Eine ähnliche Vorgehensweise wird in [dM95] beschrieben.

Die Integration von Meßwerkzeugen in 'Performance-Managementsysteme' (vgl. Abschnitt 1.3.2.7) ist im Kontext von Hochgeschwindigkeitsnetzen (HSN) besonders interessant. Durch die hohen Übertragungsgeschwindigkeiten kann ein HSN innerhalb weniger Millisekunden von einem unkritischen Zustand in einen kritischen Zustand[46] wechseln, der nur durch 'Management-Eingriffe' behoben werden kann[47]. Eine frühzeitige Erkennung kritischer Zustände ist bei HSN durch automatische Überwachung und Reaktion mit einem 'Performance-Managementsystem' möglich.

5.7.4 Vergleich mit anderen Werkzeugen der Kommunikationsmeßtechnik

Ein Überblick über Werkzeuge[48] der Kommunikationsmeßtechnik wird in RFC 1470 gegeben. Der dort vorgeschlagenen Klassifikation folgend, müßte man den Prototypen in die Kategorien *Analyser, Benchmark, Traffic, Ethernet, IP, Eavesdrop, OS/2, Standanlone, Free* einordnen.

Die Datenerfassung und die Funktionen des Prototypen zur Protokollierung von Dateneinheiten sind vergleichbar mit bekannten Werkzeugen wie beispielsweise SNIF-

[44] Die Ethernet-Schnittstelle des Meßwerkzeuges verfügt über eine eigene eindeutige Adresse.

[45] Neben der Rohdatenerfassung könnten optional Statistiken erstellt werden.

[46] Beispielsweise eine drohende Sättigung des Kommunikationssystems.

[47] Beispielsweise durch Aktivieren von Flußkontrollmechanismen zur Sättigungskontrolle.

[48] In RFC 1470 werden Hardware, Software (Programme und Funktionsbibliotheken) und hybride Realisierungen für die Kommunikationsmeßtechnik als 'Werkzeuge' bezeichnet.

FER (vgl. RFC 1470, S. 136ff). Beide Werkzeuge sind Hardware-Monitore[49], die die Dateneinheiten des Netzes auswerten, an das sie angeschlossen sind. Der Funktionsumfang des kommerziellen Werkzeuges übertrifft den hier realisierten Prototypen bei weitem.

Ein weiterer Unterschied besteht in der Granularität der Parallelverarbeitung. SNIFFER verfügt neben der Schnittstelle zum Netz nur über eine weitere Komponente, die parallel zum 'Hostrechner' arbeitet (vgl. [Hau88, S. 379]). Eine solche Architektur ist nur durch schnellere Hardware an HSN anpaßbar, jedoch nicht durch weitere Prozessoren[50].

Die Analyse erfolgt beim Prototypen grundsätzlich in Echtzeit. SNIFFER analysiert die Dateneinheiten hingegen 'off-line' (vgl. Abschnitt 1.4.5). In einem 'Trace Buffer' werden Dateneinheiten gesammelt und nach Abschluß der Rohdatenerfassung analysiert. Im Kontext von HSN ist dieses Vorgehen nicht zu empfehlen, da hierfür enorme Betriebsmittel für die Zwischenspeicherung der Rohdaten benötigt werden.

[49] Der 'Meßfühler' ist die Schnittstelle zum Kommunikationssystem. In diesem Fall eine Ethernet-Schnittstelle.

[50] Der Prototyp könnte durch weitere Transputer noch mehr Paketträger parallel einsetzen.

Kapitel 6

Arbeiten mit dem Prototypen

In diesem Kapitel soll anhand von Beispielen das Arbeiten mit dem Prototypen erklärt werden. Dieses Kapitel ist sozusagen das "Benutzerhandbuch" des Meßwerkzeuges.

Aus zeitlichen Beschränkungen der Diplomarbeit und Restriktionen der Hardware ergeben sich typische Engpässe beim Einsatz des Prototypen. Diese Engpässe werden zusammen mit Lösungsvorschlägen zu ihrer Beseitigung in Kapitel 7 aufgezeigt.

6.1 Die Benutzerschnittstelle des Prototypen

Der 'Hostrechner'-Prozeß kann durch Eingabe des Befehls 'pmfrontend' unter einem OS/2 Befehlsinterpreter gestartet werden. Alternativ kann ein 'Icon' des Programms mit der Maus 'angeklickt' werden. Vorzuziehen ist jedoch das Starten über den Befehlsinterpreter, da der Anwender sich so vergewissern kann, in welches (Datei-)Verzeichnis die Dateien geschrieben werden, die im Zuge einer Messung vom 'Hostrechner'-Prozeß angelegt werden.

Unmittelbar nach dem Start werden verschiedene 'Fenster' des Programms auf der graphischen Oberfläche dargestellt. Es handelt sich dabei um folgende Fenster (mit den Fensternamen in runden Klammern):

- Die 'Standardausgabe' (PM STDOUT). In diesem Fenster werden alle Statusnachrichten der Prozesse ausgegeben. Außerdem werden hier die Benutzereingaben protokolliert, Fehlermeldungen und Hinweise ausgegeben.
- Die 'ARP/RARP-Tabelle' (ARP RARP Tabelle). Alle Informationen aus ARP- und RARP-Nachrichten werden in diesem Fenster angezeigt, sofern der Benutzer die ARP- und RARP-Protokollanalyse einschaltet. Bei einer Erweiterung um Analysefunktionen für 'Domain Name System'-Informationen, könnte man hier zusätzlich symbolische Namen zu den Adressen anzeigen.
- Die 'Auslastungsanzeige' (MBit/s). In diesem Fenster wird die Auslastung des Ethernet-Segmentes auf den verschiedenen Protokollebenen angezeigt. Die Skalierung ist in Schritten von einem 'MBit' einstellbar. Zusätzlich kann eine Le-

gende eingeblendet werden, um die farblich markierte Auslastung den Protokollen zuordnen zu können.

- Die 'Benutzereingabe' (EINGABE:). Der Benutzer kann hier die Kommandos eintippen und gegebenenfalls korrigieren. Kommandos werden mit einem Semikolon abgeschlossen und dann an den Kommandointerpreter abgeschickt. Eingabefehler werden in dem Fenster 'Standardausgabe' angezeigt.

Mit Ausnahme des Fensters 'Benutzereingabe', das sich relativ zur 'Standardausgabe' bewegt, sind alle Fenster frei skalierbar und auf der graphischen Oberfläche frei bewegbar. Die Fenster können einzeln geschlossen und wieder geöffnet werden. In Abbildung 6.1 ist eine mögliche Anordnung der Fenster dargestellt.

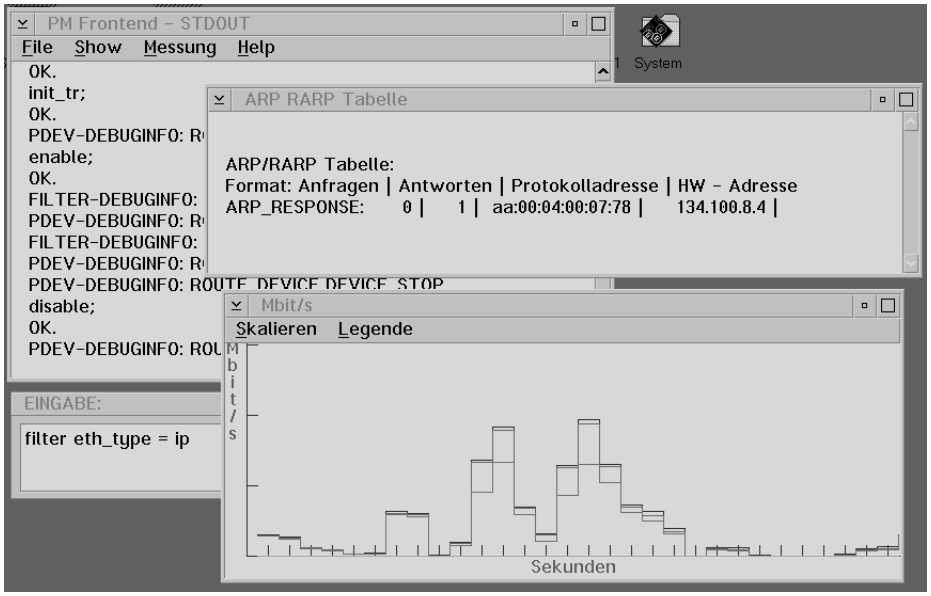


Abbildung 6.1: Fenster des 'Hostrechner'-Prozesses

6.1.1 Das Fenster 'Standardausgabe'

Der 'Hostrechner'-Prozeß versucht nach dem Start die Transputer zu konfigurieren und die Prozesse in das Transputernetz zu laden. Dieser Vorgang dauert ein paar Sekunden. In dieser Zeit kann der Benutzer keine Eingaben machen. Ist die Konfigurierung gelungen, melden sich die Prozesse des Transputernetzes mit einer Startmeldung. Der Benutzer kann so prüfen, ob alle Prozesse ordnungsgemäß geladen wurden. Die Ausgabe erfolgt im 'Standardausgabe'-Fenster wie in Abbildung 6.2 dargestellt.

Bei diesem Fenster handelt es sich um ein 'Textfenster' (vgl. Abschnitt 5.3.4.2). Man erkennt auf der rechten Seite den 'Rollbalken' zum Verschieben des sichtbaren

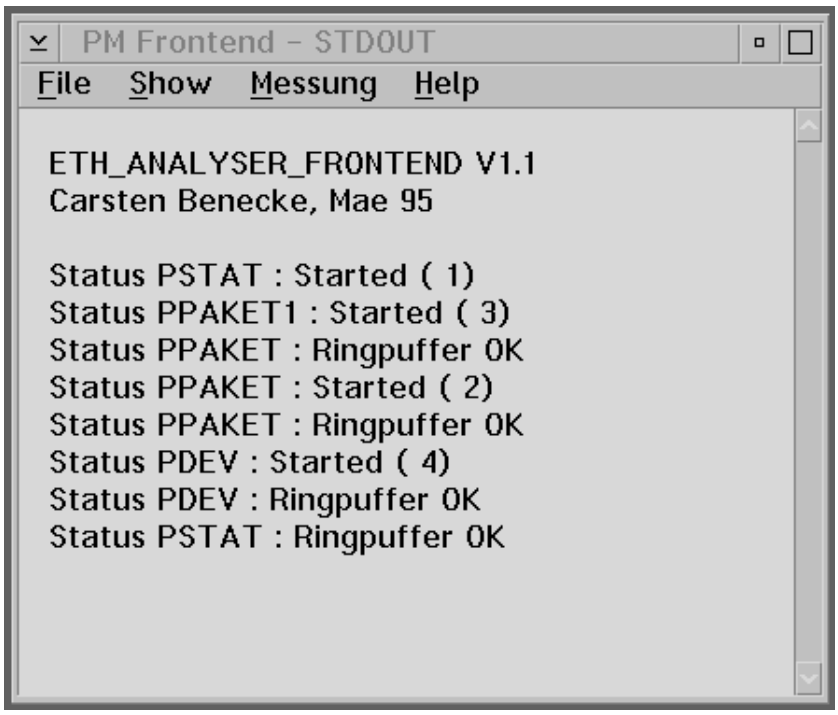

PM Frontend - STDOUT
File Show Messung Help
ETH_ANALYSER_FRONTEND V1.1
Carsten Benecke, Mae 95
Status PSTAT : Started (1)
Status PPAKET1 : Started (3)
Status PPAKET : Ringpuffer OK
Status PPAKET : Started (2)
Status PPAKET : Ringpuffer OK
Status PDEV : Started (4)
Status PDEV : Ringpuffer OK
Status PSTAT : Ringpuffer OK

Abbildung 6.2: Standardausgabe

Textausschnittes. In der rechten oberen Ecke des Fensters sieht man zwei Schalter zum Minimieren und Maximieren des Fensters[1].

Einige Kommandos können über das Menü des 'Standardausgabe'-Fensters ausgewählt werden. Die Menüeinträge werden kurz beschrieben.

6.1.1.1 Das 'File'-Menü

Über das 'File'-Menü können folgende Kommandos ausgewählt werden:

- Transputer booten. Die Prozesse werden normalerweise automatisch in das Transputernetz geladen (s.o.). Falls das Laden der Prozesse nicht gelingt, kann der Benutzer mit diesem Kommando einen weiteren Versuch zum Laden der Prozesse auslösen.
- About Eth_Analyser. Der Benutzer wird darüber informiert, daß der Prototyp im Rahmen einer Diplomarbeit erstellt wurde und den (vorläufigen) Namen

[1] Der Benutzer kann beim Konfigurieren der graphischen Benutzerschnittstelle von OS/2 festlegen, wie sich Fenster beim 'Drücken' des Schalters zum Minimieren verhalten sollen. Eine mögliche Reaktion ist, daß das Fenster zu einem Symbol ('Icon') verkleinert wird, das nur sehr wenig Platz auf der graphischen Oberfläche beansprucht. Alternativ kann das Fenster bis zu einer manuellen Reaktivierung vollständig von der Oberfläche gelöscht werden.

'Eth_Analyser' trägt.

- Fenster speichern und Fenster laden. Die Fenster werden beim Programmstart von OS/2 (willkürlich) auf der graphischen Oberfläche plaziert. Damit der Benutzer nicht nach jedem Programmstart das 'Layout' der Fenster mit der Maus modifizieren muß, können individuelle Präferenzen des Benutzers zur Lage und Größe der Fenster gespeichert und geladen werden.
- Exit. Das Beenden des Programms ist nur erlaubt, falls keine Messung aktiviert ist (vgl. Abschnitt 6.2).

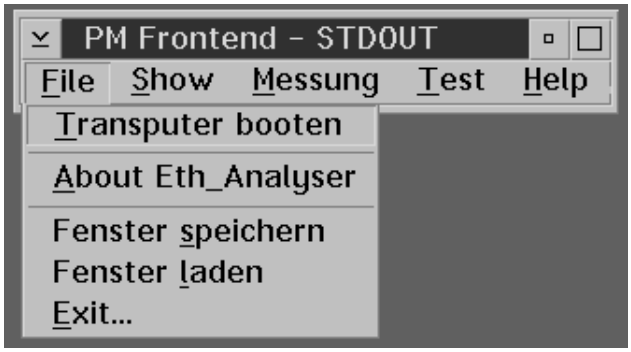


Abbildung 6.3: Das 'File'-Menü

6.1.1.2 Das 'Show'-Menü

Über dieses Menü kann der Benutzer Informationen abrufen:

- Aktuelle Messung. Zeigt den Namen und den Zustand der Messung an (aktiviert vs. unterbrochen, s.u.).
- Prozesse. Es wird eine 'Broadcast'-Nachricht an alle Transputerprozesse gesendet. Die Prozesse antworten mit einer Zustandsmeldung. Der Benutzer kann durch diese Funktion überprüfen, ob alle Prozesse auf Nachrichten reagieren.
- Netz. Es werden die netznahen Statistiken angefordert und ausgegeben (vgl. Abschnitt 4.4.1).
- ARP Tabelle. Das Fenster der ARP-Tabelle wird wieder hergestellt bzw. vor allen anderen Fenstern angezeigt. Typischerweise wird das ARP-Fenster von der graphischen Oberfläche gelöscht, da es nur bei Messungen benötigt wird, in deren Verlauf ARP/RARP-Dateneinheiten analysiert werden. Über diesen Menüeintrag kann das temporär gelöschte Fenster wieder hergestellt werden.

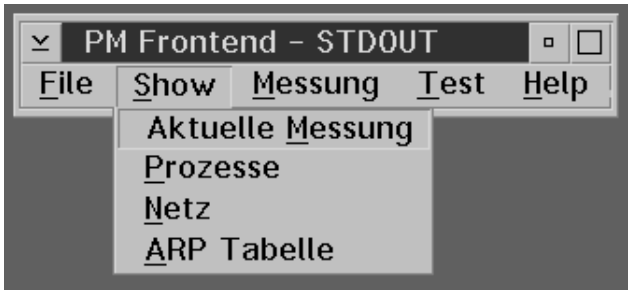


Abbildung 6.4: Das 'Show'-Menü

6.1.1.3 Das 'Messung'-Menü

Die wichtigsten Kommandos zur Steuerung einer Messung sind hier zusammengefaßt:

- Aktuelle Messung. Dieser Eintrag entspricht dem Eintrag im 'Show'-Menü.
- Starten. Eine Messung wird aktiviert (s.u.).
- Unterbrechen. Eine Messung wird unterbrochen (s.u.).
- Parameter festlegen. Ein weiteres Fenster, mit Dialogen zum Auswählen der zu analysierenden Protokolle, wird abgerufen (s.u.).

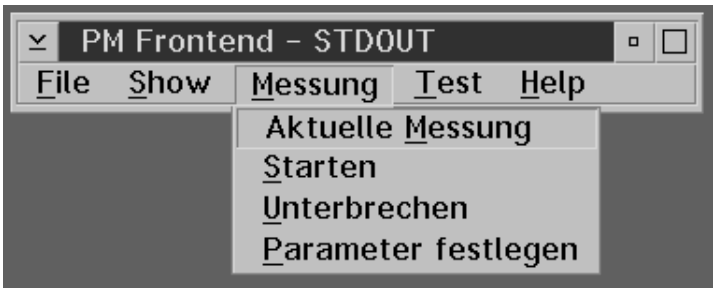


Abbildung 6.5: Das 'Messung'-Menü

6.1.1.4 Andere Menüs

Die Einträge des Hilfe-Menüs lösen keine Kommandos aus, da bisher keine Hilfsfunktionen realisiert wurden (vgl. Abschnitt 5.7.2). Die Einträge im Test-Menü werden nicht zur Durchführung von Messungen benötigt. Es können Kommandos zur Leistungsbewertung einzelner Komponenten ausgelöst werden (vgl. Kapitel 7).

6.1.2 Das Fenster 'Standardeingabe'

In dem Fenster 'Standardeingabe' wird die Kommandozeile (vgl. Abschnitt 5.3.4.1) dargestellt. Bei sehr langen Eingaben kann der Benutzer den Text mit dem 'Rollbalken' verschieben. Am Zeilenende (rechter Fensterrand) wird das aktuelle Wort automatisch in die folgende Zeile verschoben. Abbildung 6.6 zeigt das Fenster mit einer unvollständigen Benutzereingabe in der Kommandozeile.

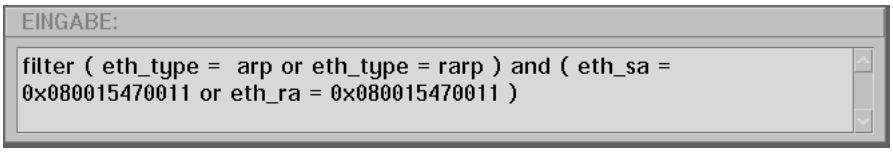


Abbildung 6.6: 'Standardeingabe'-Fenster

6.1.3 Das Fenster 'MBits/s'

In diesem Fenster wird die Auslastung des Ethernet-Segmentes in Megabit pro Sekunde angezeigt. Auf der X-Achse sind 30 Sekunden abgetragen. Die Y-Achse läßt sich von ein bis 11 Megabit skalieren. Die Auslastung durch Dateneinheiten der verschiedenen Protokolle wird als Histogramm dargestellt. Die Kurven der Protokolle werden mit verschiedenen Farben dargestellt. Optional kann eine Legende eingeblendet werden, die die Zuordnung von Protokollen zu Farben angibt. Ein möglicher Kurvenverlauf wird in Abbildung 6.11 dargestellt.

Auch dieses Fenster ist frei skalierbar. Da für die Darstellung des Koordinatensystems, der Legende und der Kurven eine Mindestgröße erforderlich ist, wird bei Unterschreiten dieser Mindestgröße eine Fehlermeldung anstelle des Histogramms im Fenster angezeigt (vgl. Abbildung 6.7).

Abbildung 6.7: Ein zu kleines Fenster

6.1.4 Das Fenster 'ARP/RARP-Tabelle'

Ein weiteres Textfenster[2] wird benutzt, um die Zuordnung von Ethernet-Adressen zu IP-Adressen darzustellen. Diese Informationen ergeben sich aus der Analyse der Protokolle ARP und RARP (vgl. Abschnitt 5.3.2.3). Für jedes Adreßpaar wird notiert, wie oft es Anfragen und Antworten bezüglich einer Adresse gab. Bei der Ausgabe im Fenster werden die Adressen, die zuletzt in einer ARP/RARP-Dateneinheit empfangen wurden, zuerst ausgegeben ('lifo'-Prinzip). Abbildung 6.10 zeigt Beispiele für Adreßpaare.

6.1.5 Fehler beim Programmstart

Tritt während des Konfigurierens der Transputer ein Fehler auf, wird das vom 'Hostrechner'-Prozeß bemerkt und in einem 'Dialog-Fenster'[3] angezeigt. Außerdem wird angegeben, bei welchem der beiden 'Transputerboards' das Problem auftritt.

Es gibt vier wahrscheinliche Fehlerursachen:

1. Ein 'Transputerboard' ist nicht eingesetzt oder für einen anderen I/O-Bereich konfiguriert. Dieser Fehler tritt auf, falls die 'Transputerboards' (TBs) nicht richtig im 'Hostrechner' eingesetzt wurden, oder über die Steckleisten ein anderer I/O-Bereich für die Adressen der TBs gewählt wurde. Die zu verwendende Startadresse für das Parsytec-'Board' (vgl. Abbildung 3.7) ist 150_{16}. Für das INMOS-'Board' (vgl. Abbildung 3.8) ist die Startadresse 300_{16} zu wählen.

2. Eine falsche Konfigurationsdatei wurde beim Konfigurieren der Transputer-Prozesse verwendet. Es muß überprüft werden, ob in den Programmquellen der richtige Dateiname eingetragen wurde.

3. Eine falsche Programmdatei mit den Transputerprozessen wurde angegeben. Auch hier ist die Angabe in den Programmquellen der Ladefunktion (vgl. Abschnitt 5.3.4.6) zu überprüfen.

4. Der 'Timeout'-Wert für den Ladevorgang wurde zu klein gewählt (vgl. Abschnitt 5.3.4.6). Dieser Wert kann in den Programmquellen versuchsweise erhöht werden.

6.2 Vorgehensweise zur Durchführung eines Meßexperiments

Allgemein sind folgende Schritte für die Durchführung eines Meßexperiments notwendig:

[2] Die Funktionalität entspricht dem 'Standardausgabe'-Fenster.

[3] OS/2 stellt für Situationen wie diese, wo der Benutzer in jedem Fall eine Meldung quittieren sollte, sogenannte 'system-modale Dialoge' zur Verfügung. Neben dem Anzeigen der (Fehler-)Meldung und auffälliger 'Achtung'-Symbole blockiert ein solcher Dialog alle Eingaben, bis der Benutzer den Dialog bestätigt.

- Benennen der Messung,
- Definieren eines Filters,
- Auswählen der zu protokollierenden Informationen,
- Durchführen der Messung,
- Beenden einer Messung.

6.2.1 Benennen der Messung

Jeder Messung muß ein Name gegeben werden. Dieser Name wird benutzt, um Betriebsmittel für die Messung zu verwalten. Außerdem können über den Namen Informationen über die Messung abgerufen werden, beispielsweise ob gerade Meßdaten erhoben werden.

6.2.2 Definieren eines Filters

Man benötigt keinen Filter für die Durchführung einer Messung. Ohne Filter werden alle Rohdaten analysiert. Man sollte jedoch immer versuchen, einen adequaten Filter zu definieren. Wichtig ist das Filtern bei langen Messungen, die protokolliert werden, um möglichst nur Meßwerte zu protokollieren, die tatsächlich benötigt werden. Bei sehr aufwendigen Analysen kann es u.U. notwendig sein, Filter zu definieren, um die Last an die Paketträger zu verringern.

6.2.3 Auswählen der zu protokollierenden Informationen

Hier können protokollspezifische Angaben gemacht werden, welche Parameter zu protokollieren sind, falls eine Dateneinheit den Filter passiert. Es handelt sich hierbei um einen zweiten Filter. Beispielsweise kann der Benutzer angeben, daß bei der Analyse von IP-Dateneinheiten die Länge der Dateneinheit ausgegeben wird aber die Sender- und Empfängeradressen ignoriert werden können.

6.2.4 Durchführen der Messung

Die Meßdatenerfassung wird aktiviert. Es werden so lange Meßdaten erhoben, bis der Benutzer die Messung unterbricht. Nach einer Unterbrechung kann die Messung erneut gestartet oder beendet werden. Der Benutzer unterbricht eine Messung in der Regel, um einen neuen Filter einzugeben oder um die Messung zu beenden.

6.2.5 Beenden einer Messung

Nach dem Unterbrechen kann der Benutzer eine Messung beenden. Dadurch werden alle Betriebsmittel, die für die Messung reserviert wurden, wieder freigegeben. Der Name der Messung ist nach dem Beenden nicht mehr gültig[4]. Nach dem Beenden einer

[4] Der Name kann für eine weitere Messung erneut vergeben werden.

Messung kann der Benutzer das Programm beenden. Während der Meßdatenerfassung ist das Beenden des Programms nicht erlaubt. Ein Versuch, das Programm während der Meßdatenerfassung zu beenden, wird mit einer Fehlermeldung vom Prototypen abgelehnt[5].

In den folgenden Abschnitten werden zwei Fallbeispiele für die Durchführung von Meßexperimenten gegeben. In Abschnitt 6.3 werden ARP/RARP-Dateneinheiten ausgewertet. In Abschnitt 6.4 wird eine Lastmessung durgeführt.

6.3 Fallbeispiel I: Analysieren von ARP/RARP-Dateneinheiten

Das folgende Protokoll einer Messung zeigt, wie das Analysieren von ARP/RARP-Dateneinheiten durchgeführt werden kann.

```
 1 OK.
 2 status process;
 3 OK.
 4 Status PSTAT : OK
 5 Status PPAKET : OK
 6 Status PPAKET : OK
 7 Status PDEV : OK, nicht gebootet, nicht terminiert,
                          Empfang nicht eingeschaltet.
 8 PDEV-DEBUGINFO: ROUTE_BROADCAST SI
 9 init_tr;
10 OK.
11 PDEV-DEBUGINFO: ROUTE_DEVICE STARTUP
12 mess _arp;
13 OK.
14 filter eth_type = arp or eth_type =rarp;
15 SYNTAX FEHLER!!
16 filter eth_type = arp or eth_type = rarp;
17 OK.
18 enable;
19 OK.
20 PDEV-DEBUGINFO: ROUTE_DEVICE NEW_FILTER
21 FILTER-DEBUGINFO: NEW_FILTER
22 FILTER-DEBUGINFO: RESET
23 PDEV-DEBUGINFO: ROUTE_DEVICE DEVICE_START
24 disable;
25 OK.
```

[5]Der Benutzer kann — bei entsprechender Kenntnis des Betriebssystems — das Programm durch ein (programm-)externes Kommando beenden. Dieses Vorgehen ist nur sinnvoll, falls der Prototyp nicht mehr auf Kommandos reagiert.

```
26 PDEV-DEBUGINFO: ROUTE_DEVICE DEVICE_STOP
27 status;
28 _arp   Messung DISABLED
29 OK.
30 drop filter;
31 OK.
32 drop _arp;
33 OK.
34 spool off;
```

Dieses Protokoll wurde von einer Funktion erstellt, die mit dem Befehl 'spool dateiname;' alle Ausgaben in das 'Standardausgabe'-Fenster zusätzlich in eine Datei (der Name wird anstelle von 'dateiname' angegeben) schreibt. Das 'OK.' in Zeile 1 ist die Bestätigung für dieses Kommando.

Zu jeder Zeit kann der Benutzer Statusinformationen von den Prozessen erfragen. Der Benutzer kann so feststellen, ob die Prozesse noch erreichbar sind. Dazu wird das Kommando 'status process;' (in Zeile 2) eingegeben. Während sich die Paketträger und die globale Statistik nur mit einem 'OK' melden (Zeile 4–6), gibt der Prozeß 'PDEV' etwas mehr Auskunft (Zeile 7). Er meldet, daß der Netz-'Server' bisher nicht 'gebootet' wurde. Die beiden anderen Informationen besagen, daß der Netz-'Server' noch nicht terminiert wurde und der Paketempfang des 'Servers' nicht eingeschaltet wurde.

Zur Zeit muß der Prototyp noch vom Benutzer aufgefordert werden, den Netz-'Server' zu laden[6]. Dazu gibt der Benutzer 'init_tr;' ein (Zeile 9). Ein weiteres Kommando 'status process;' würde dann den Zustand 'PDEV OK, gebootet,...' melden.

Beim Prototypen werden darüber hinaus 'Debug'-Informationen von den Prozessen generiert, falls der Benutzer das wünscht. In diesem Fall generiert der Prozeß 'PDEV' solche Informationen. Er zeigt die Nachrichten und den Typ der Vermittlungsinformationen der Nachrichten an, die er empfängt[7].

6.3.1 Name der Messung

Jede Messung muß einen Namen bekommen. In diesem Fall vergibt der Benutzer in Zeile 12 den Namen '_arp' mit dem Kommando 'mess _arp;'. Der Name erfüllt zwei Aufgaben. Einerseits kann der Benutzer sich auch nach Tagen anhand des Namens daran erinnern, was er gemessen hat (bzw. was er messen wollte). Andererseits werden Namen für Protokollierungs-Dateien und Meßergebnisse über den Namen der Messung verwaltet (s.u.).

[6] Der Grund dafür ist, daß ein Großteil der Konfiguration getestet werden kann, ohne den 'Server' zu laden. Für diese Testzwecke konnten ältere Versionen des Prototypen auch ohne Ethernet-Schnittstelle betrieben werden. Bei einer zukünftigen Version sollte das Laden des Netz-'Server' jedoch automatisch mit dem Laden der anderen Prozesse erfolgen.

[7] In Zeile 8 informiert 'PDEV' den Benutzer, daß er eine Nachricht vom Typ 'SI' (Status Information) durch eine 'Broadcast'-Nachricht (ROUTE_BROADCAST) erhalten hat. Diese Nachricht hat das Versenden der Informationen aus Zeile 7 veranlaßt.

6.3.2 Definieren eines Filters

Man sollte in jedem Fall einen Filter setzen, der genau angibt, welche Dateneinheiten für die Messung analysiert werden sollen. In diesem Fall sind das ARP- und RARP-Dateneinheiten. Der Benutzer definiert einen entsprechenden Filter in Zeile 16. Der Versuch in Zeile 14 erzeugte eine Fehlermeldung (Zeile 15), da ein Leerzeichen zwischen dem relationalen Operator '=' und dem Vergleichswert (rarp) fehlt (vgl. Abschnitt 5.3.5.1).

6.3.3 Auswählen der zu protokollierenden Informationen

Über ein Dialogfeld legt der Benutzer fest, welche Informationen aus den Dateneinheiten selektiert werden sollen, die den Filter passieren. In Abbildung 6.8 werden dazu vom Benutzer die Protokolle ARP/RARP ausgewählt und angegeben, daß die Informationen der ARP/RARP-Dateneinheiten protokolliert werden sollen (vgl. Abschnitt 5.3.4.3).

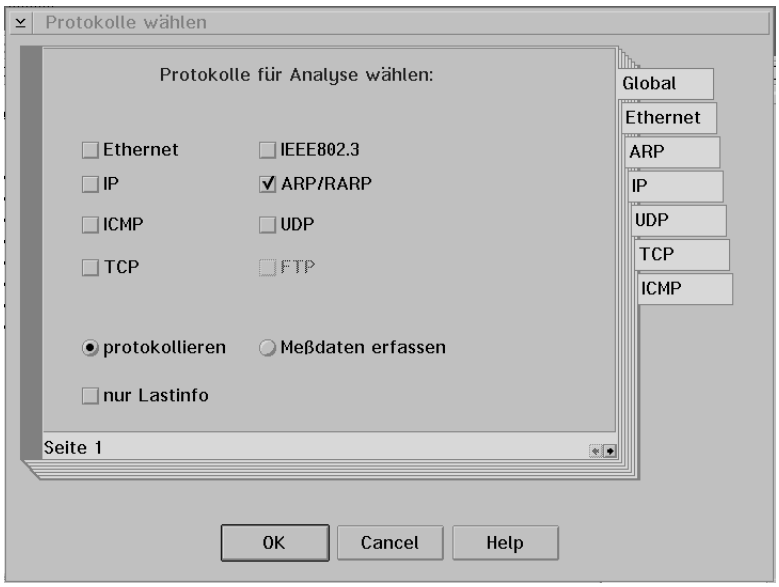


Abbildung 6.8: Protokolle wählen

Zu jedem Protokoll kann der Benutzer außerdem angeben, welche Informationen protokolliert werden sollen (vgl. Abbildung 6.9). In diesem Fall ist das unnötig, da die Angaben nur im 'ARP/RARP Fenster' angezeigt werden sollen und die dafür benötigten Informationen dem Programm bekannt sind.

Hätte der Benutzer beispielsweise mit 'set output arp.out;' eine Protokollierungsdatei (hier 'arp.out') angegeben, würden die in Abbildung 6.9 selektierten Informationen zusätzlich in dieser Datei protokolliert. Wie aus Abbildung 6.9 ersichtlich ist, können

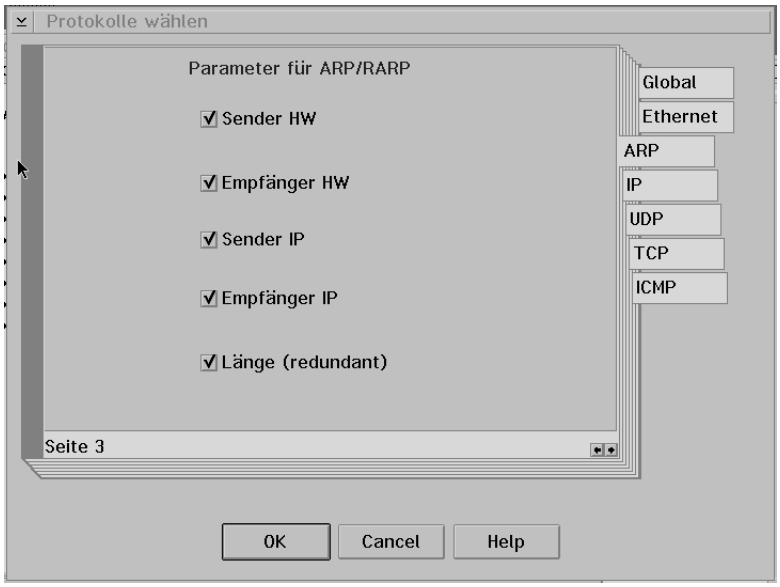


Abbildung 6.9: Zu protokollierende Informationen wählen

ähnliche Angaben für die anderen analysierbaren Protokolle gemacht werden. Wird das Werkzeug um Analysefunktionen für andere Protokolle erweitert, müssen entsprechende Wahlmöglichkeiten für diese Protokolle vorgesehen werden.

6.3.4 Starten einer Messung

Zum Starten der Messung wird das Kommando 'enable;' eingegeben (Zeile 18). Der 'PDEV'-Prozeß gibt in den Zeilen 20–23 mehrere 'Debug'-Informationen aus, die unter anderem besagen, daß der 'P-Code' des Filters empfangen wurde (vgl. Abschnitt 5.3.5.2).

Der Netz-'Server' beginnt mit dem Empfang der Dateneinheiten. Werden ARP- oder RARP-Dateneinheiten empfangen, stellt der 'Hostrechner'-Prozeß diese tabellarisch in dem Fenster 'ARP/RARP Tabelle' dar. Zusätzlich wird angezeigt, wie oft nach einer Adresse 'gefragt' wurde und wie oft eine ARP-Antwort für diese Adresse empfangen wurde. Ein Beispiel für die Ausgabe im Fenster ist in Abbildung 6.10 dargestellt.

6.3.5 Unterbrechen einer Messung

Eine Messung wird durch das Kommando 'disable;' (Zeile 24) unterbrochen. Man muß Messungen unterbrechen, um neue Filter zu definieren. Man kann Messungen unterbrechen, um beispielsweise Ausgaben in ein Fenster zu verhindern ('ARP/RARP Tabelle' oder 'MBit/s'). Das ist beispielsweise sinnvoll, um sich Ausgaben in den Fenstern 'in Ruhe' ansehen zu können.

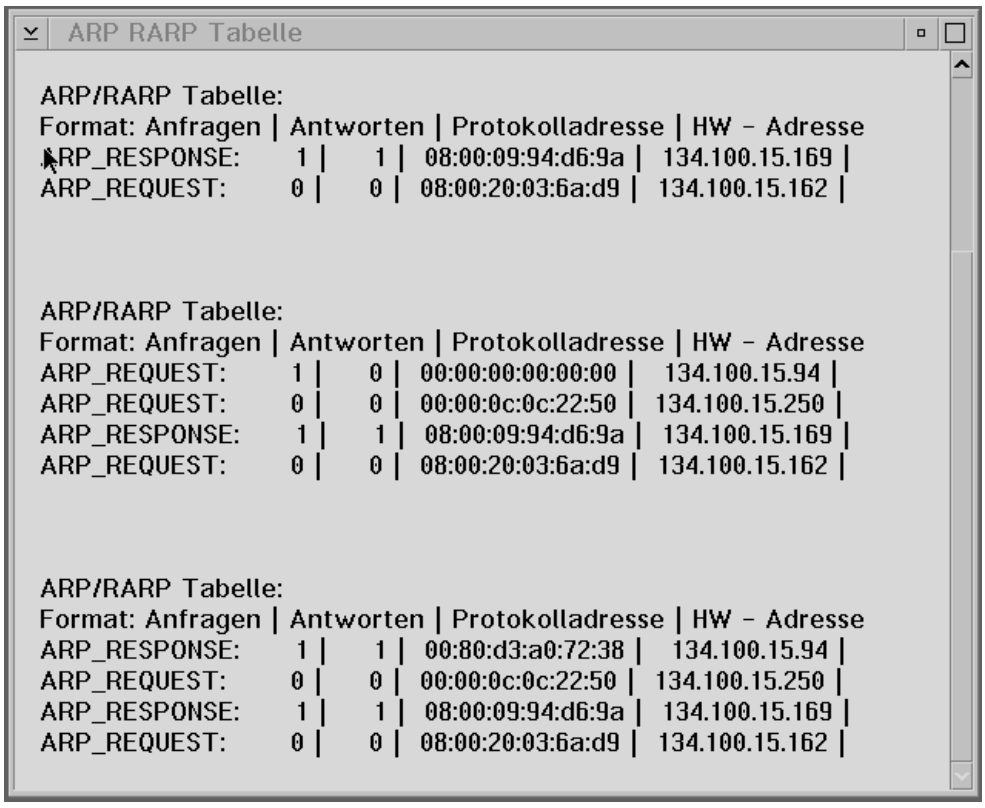

ARP RARP Tabelle
ARP/RARP Tabelle:
Format: Anfragen | Antworten | Protokolladresse | HW - Adresse
ARP_RESPONSE: 1 | 1 | 08:00:09:94:d6:9a | 134.100.15.169 |
ARP_REQUEST: 0 | 0 | 08:00:20:03:6a:d9 | 134.100.15.162 |
ARP/RARP Tabelle:
Format: Anfragen | Antworten | Protokolladresse | HW - Adresse
ARP_REQUEST: 1 | 0 | 00:00:00:00:00:00 | 134.100.15.94 |
ARP_REQUEST: 0 | 0 | 00:00:0c:0c:22:50 | 134.100.15.250 |
ARP_RESPONSE: 1 | 1 | 08:00:09:94:d6:9a | 134.100.15.169 |
ARP_REQUEST: 0 | 0 | 08:00:20:03:6a:d9 | 134.100.15.162 |
ARP/RARP Tabelle:
Format: Anfragen | Antworten | Protokolladresse | HW - Adresse
ARP_RESPONSE: 1 | 1 | 00:80:d3:a0:72:38 | 134.100.15.94 |
ARP_REQUEST: 0 | 0 | 00:00:0c:0c:22:50 | 134.100.15.250 |
ARP_RESPONSE: 1 | 1 | 08:00:09:94:d6:9a | 134.100.15.169 |
ARP_REQUEST: 0 | 0 | 08:00:20:03:6a:d9 | 134.100.15.162 |

Abbildung 6.10: ARP/RARP-Fenster

Ein Seiteneffekt beim Unterbrechen und Neustarten ('enable;') einer Messung ist, daß die Zeitstempel in einer Protokollierungsdatei wieder bei Null beginnen. Dadurch erkennt man auch nachträglich, daß eine Messung unterbrochen wurde.

Das Unterbrechen einer Messung ist außerdem Voraussetzung für das Beenden einer Messung (s.u.).

6.3.6 Beenden einer Messung

Man beendet eine Messung durch Eingabe des Kommandos 'drop name;'. 'name' ist dabei durch den Namen der jeweiligen Messung zu ersetzen. In diesem Fall gibt der Benutzer 'drop _arp;' ein (Zeile 32). Das Kommando schließt im wesentlichen die Ausgabedateien, die im Zuge der Messung angelegt wurden. Bei Erweiterungen sollten an dieser Stelle alle Funktionen aufgerufen werden, die ein 'ordentliches' Freigeben von Betriebsmittel veranlassen, die für die Messung angefordert wurden[8].

Zuvor gibt der Benutzer in Zeile 30 den Befehl 'drop filter;' ein. Bei einem Neustart der Messung werden dann wieder alle Dateneinheiten ausgewertet. Ein Filter bleibt

[8] In diesem Fall sind das Einträge in der Symboltabelle, die mit dem Namen '_arp' verbunden waren.

immer solange erhalten, bis er vom Benutzer explizit gelöscht wird.

Das Kommando 'spool off' (Zeile 34) unterbricht das Protokollieren der 'Standardausgabe' in eine Datei. Die Kommandos 'show' und 'spool' sind grundsätzlich unabhängig von einer Messung. Sie können zu jeder Zeit eingegeben werden, also auch während einer Messung.

6.4 Fallbeispiel II: Durchführen einer Lastmessung

Als zweites Beispiel wird eine Lastmessung durchgeführt. Durch eine Lastmessung kann man sich anzeigen lassen, wie groß die Auslastung des Ethernet-Segmentes ist und durch welche Art von Aufträgen die Last erzeugt wird.

6.4.1 Auswählen der anzuzeigenden Protokolle

Es soll die Auslastung des Ethernet-Segmentes während der Messung graphisch dargestellt werden. Dabei wird analog zum obigen Beispiel vorgegangen. Der Messung wird ein Name gegeben und es wird gegebenenfalls ein Filter gesetzt (z.B. um nur die Last zu messen, die durch einen bestimmten Knoten oder eine spezielle Verbindung in das Netz eingebracht wird).

Bei der Auswahl der Protokolle 'kreuzt' man im Dialogfenster (vgl. Abbildung 6.8) alle Protokolle an, die angezeigt werden sollen. Zusätzlich sollte 'nur Lastinfo' ausgewählt werden. Dadurch werden alle Analyseergebnisse, die nicht für die Berechnung der Auslastung benötigt werden, bereits im Transputernetz vernichtet. Man erreicht so, daß auch bei hoher Auslastung keine Dateneinheiten aufgrund von 'Staus' am primären Engpaß ('Transputerboard' zum 'Hostrechner') verlorengehen (vgl. Abschnitt 5.2.5.1).

6.4.2 Das Fenster 'MBit/s'

Nach dem Starten der Messung wird die Auslastung im Fenster 'MBit/s' in Megabit pro Sekunde angezeigt. Es wird grundsätzlich ein Zeitintervall von 30 Sekunden dargestellt. Die verschiedenen Protokolle sind farblich markiert. Über den Menüeintrag 'Legende' kann der Benutzer eine Legende ein- und ausblenden, die die Zuordnung von Farben zu Protokollen verdeutlicht.

Bei geringer Last ist es u.U. notwendig, den Bereich zu skalieren, um die verschiedenen Protokolle unterscheiden zu können. Der Benutzer kann dazu über den Menüeintrag 'Skalieren' den Bereich der y-Achse in Schritten von einem Megabit skalieren. Das Skalieren kann während, vor oder nach einer Messung erfolgen. Der Benutzer kann das Fenster außerdem in der Größe verändern.

Abbildung 6.11 zeigt das Fenster 'MBit/s' mit eingeblendeter Legende. Die y-Achse ist auf ein Megabit skaliert. Die Auslastung ist demnach gering.

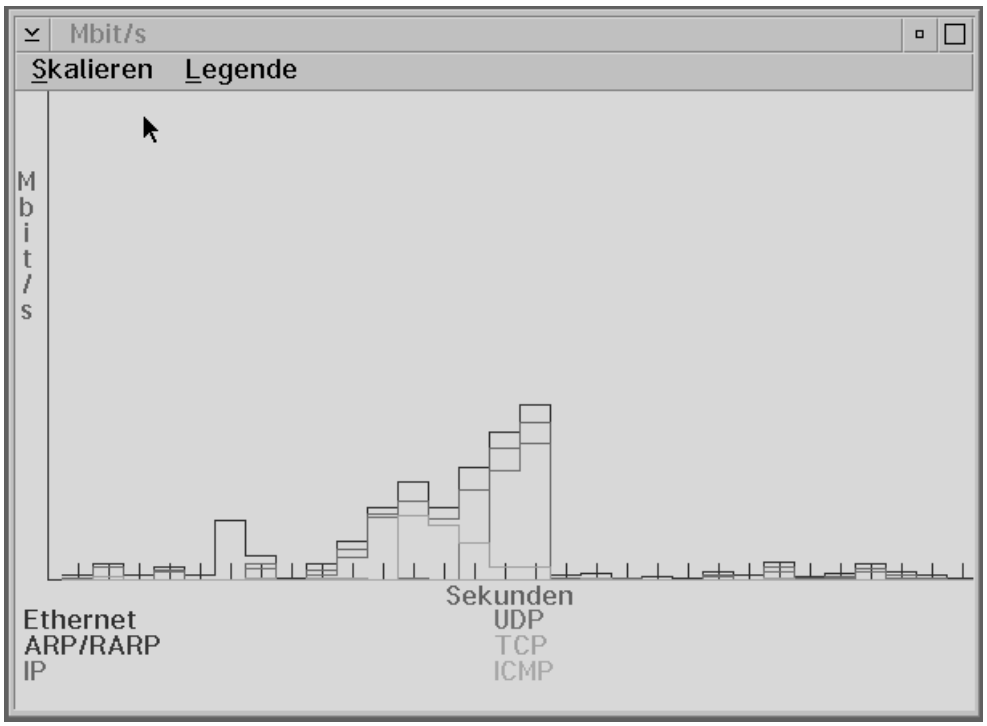


Abbildung 6.11: Fenster 'MBit/s'

Kapitel 7

Leistungsbewertung des Prototypen

In diesem Kapitel wird die Leistungsfähigkeit des Prototypen bewertet. In Abschnitt 7.1 werden zwei Engpässe identifiziert. Weitere Komponenten lassen sich unter geeigneten Annahmen bewerten. In Abschnitt 7.2 wird der Grenzdurchsatz[1] für einen 'Paketträger' und für das Filtern berechnet.

7.1 Engpässe des Prototypen

Beim Arbeiten mit dem Prototypen kann es passieren, daß der Benutzer eine Warnung im 'Standardausgabe'-Fenster erhält. Es gibt zwei Warnungen, die unabhängig voneinander auftreten können. Bei der einen Warnung wird dem Benutzer mitgeteilt, daß Analysedaten möglicherweise aufgrund eines 'Nachrichtenstaus' an der Schnittstelle zum 'Hostrechner' verlorengehen[2]. Die Ursachen hierfür werden in Abschnitt 7.1.1 diskutiert.

Die zweite Warnung zeigt an, daß möglicherweise Ethernet-'Frames' von der Ethernet-Schnittstelle vernichtet werden, weil die Auslastung der Schnittstelle zu hoch ist. Die Ursache für diese Warnung wird in Abschnitt 7.1.4 dargestellt.

Wenn eine dieser Meldungen auftritt, sind die Analyseergebnisse vermutlich falsch oder unvollständig. Der Benutzer sollte deshalb prüfen, ob die aktuelle Messung wiederholt werden kann.

[1] vgl. [JV87, S. 67]

[2] 'PSTAT' leitet die Daten zum 'Hostrechner' weiter. Eine Warnung wird von 'PSTAT' generiert, falls der Nachrichtenringpuffer voll ist. 'PSTAT' kann aber nicht definitiv entscheiden, ob das zum Verlust von Dateneinheiten führt. Die Nachrichtenringpuffer der Paketträger können beispielsweise kurzfristig Nachrichten zwischenspeichern. Falls kein Prozeß weitere Nachrichten zwischenspeichern kann, kommt es auf der Ethernet-Schnittstelle zu Paketverlusten. Dieser Paketverlust kann durch Anfordern der netznahen Statistiken erkannt werden.

7.1.1 Der primäre Engpaß des Prototypen

Der primäre Engpaß des Prototypen ist die Schnittstelle zwischen 'Hostrechner' und 'Transputerboard'. Das Datentor auf dem 'Transputerboard' ist nur acht Bit breit, obwohl die Transputer als auch der Prozessor des 'Hostrechners' 32-Bit-Prozessoren sind. Die Umwandlung der Informationen aus dem seriellen Übertragungsformat der Transputer (vgl. Abbildung 3.6) in ein acht-Bit-Wort des 'Hostrechners' ist außerdem deutlich langsamer als die Übertragungsgeschwindigkeit eines Transputerlinks. Das hat folgende drei Konsequenzen:

1. Es gibt einen deutlich geringeren Durchsatz (in Byte) zwischen 'Hostrechner' und Transputer als zwischen zwei Transputern. Das liegt an der zu langsamen Umwandlung der bit-seriell übertragenen Informationen in acht-Bit-Wörter und vice versa.

2. Das Übertragen der Daten von und zur Schnittstelle über den Bus des 'Hostrechners' ist mindestens vier mal langsamer als beim Zugriff auf andere Peripherie. Das liegt an dem nur acht Bit breiten Datentor.

3. Die Belastung des 'Hostrechner'-Prozessors ist während der Datenübertragung von und zur Schnittstelle sehr hoch. Die Ursache hierfür ist, daß es sich nicht um eine 'Direct Memory Access' (DMA)-Schnittstelle handelt, die den Prozessor von der Datenübertragung fast völlig befreit, sondern dieser jedes Byte einzeln kopieren muß. Außerdem ist bei diesem Kopiervorgang für jedes Byte die Sende- bzw. Empfangsbereitschaft der Schnittstelle zu überprüfen ('polling').

7.1.2 Analyse des primären Engpasses

Dieser Engpaß kann zu Datenverlust führen, wenn die Nachrichten zum 'Hostrechner' auf 'PSTAT' den Nachrichtenringpuffer gefüllt haben und durch sukzessives Auffüllen der Nachrichtenpuffer der anderen Prozesse ('Rückstau') letztendlich beim Netz-'Server' Dateneinheiten abgewiesen werden[3].

Durch die netznahen Statistiken, die der Benutzer mit dem Kommando 'status net;' direkt vom Netz-'Server' anfordern kann, kann man feststellen, ob Dateneinheiten abgewiesen werden mußten. Hiermit konnte gemessen werden, bei welcher Art von Aufträgen es zu einem Rückstau an der Schnittstelle zum 'Hostrechner' kommt.

In Tabelle 7.1 sind die verschiedenen Aufträge aufgelistet, die von der 'Funktionseinheit' "Schnittstelle zum 'Hostrechner' " verarbeitet werden müssen. Es sind nicht alle Aufträge, die auftreten können berücksichtigt, sondern nur solche, die mit einem externen Lastgenerator systematisch erzeugt werden konnten. Bei den Aufträgen handelt es sich um Analysedaten (für die in der Tabelle angegebenen Protokolle), die zum 'Hostrechner'-Prozeß übertragen werden sollen.

[3] Alle anderen Prozesse synchronisieren sich über die vollen Nachrichtenringpuffer. Sie warten mit der Weiterverarbeitung, bis wieder Nachrichten versendet werden können. Hier gehen jedoch auf keinen Fall Nachrichten verloren. Der Netz-'Server' kann bei voller Empfangswarteschlange allerdings keine Dateneinheiten mehr empfangen. Treffen dennoch Dateneinheiten ein, werden sie abgewiesen

Auftragsnummer	Analysedaten	Länge in Byte
1	Ethernet	68
2	Ethernet + IP	100
3	Ethernet + IP + UDP	116
4	UDP	16
5	IP + UDP	48
6	Ethernet + UDP	84
7	IP	32

Tabelle 7.1: Die verschiedenen Aufträge für die Schnittstelle zum 'Hostrechner'

In Abbildung 7.1 ist der Grenzdurchsatz in Abhängigkeit von der Auftragsgröße (Länge in Byte) dargestellt. Erwartungsgemäß werden die kürzesten Aufträge auch am schnellsten verarbeitet.

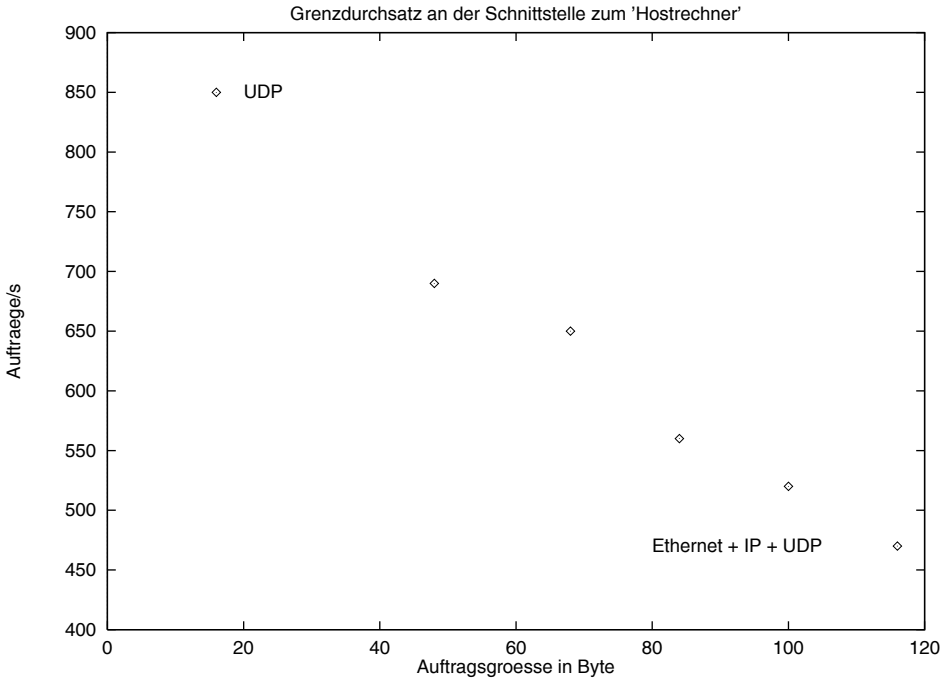


Abbildung 7.1: Grenzdurchsatz in Aufträge/s

Eine andere Sicht auf die Daten zeigt Abbildung 7.2. Der Durchsatz in Kilobyte pro Sekunde nimmt mit der Auftragsgröße zu. Das liegt an den Vorlaufzeiten, die bei jedem Kommunikationsvorgang über einen Transputerlink als 'Overhead' anfallen (vgl. [Ben94, S. 112]). Je kleiner der Auftrag, desto mehr Aufträge erzeugen im

und stehen nicht für die Analyse zur Verfügung.

Beobachtungsintervall (hier eine Sekunde) 'Overhead' durch die Vorlaufzeiten.

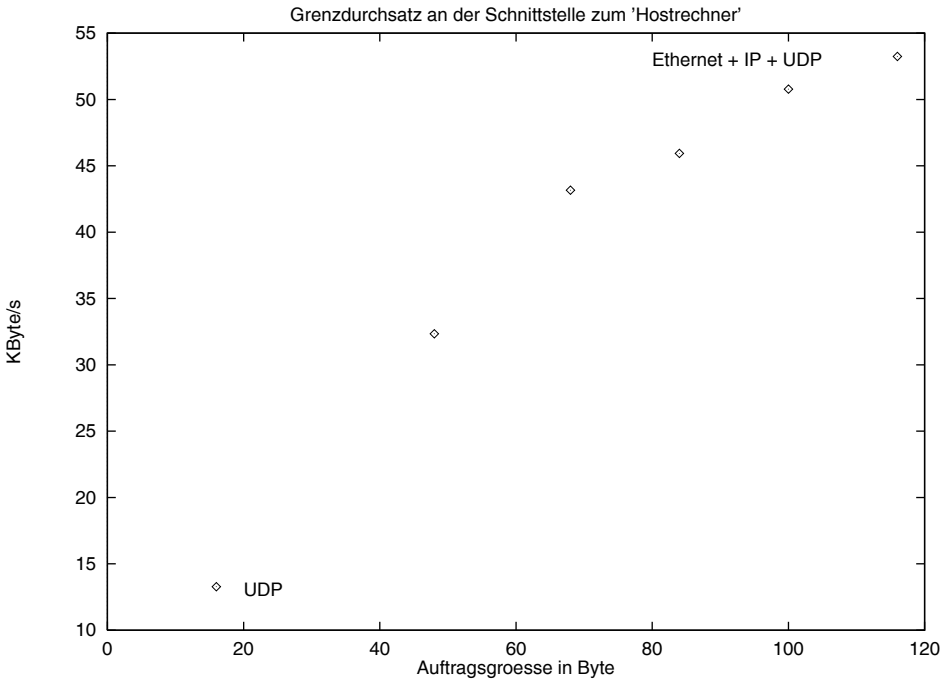


Abbildung 7.2: Grenzdurchsatz in KByte/s

7.1.3 Vorbeugende Maßnahmen

Der Benutzer sollte möglichst restriktive Filter definieren, die die Menge der zu analysierenden Dateneinheiten weitgehend reduzieren. Besonders nützlich sind hier Filter, die die Anzahl der beobachteten Knoten verringern.

Auf der zweiten Filterstufe sollten nur die Protokolle ausgewählt werden, die tatsächlich für die Meßaufgabe analysiert werden müssen. Je mehr Protokolle ausgewählt werden, desto mehr Analysedaten werden erzeugt und desto kleiner ist der Grenzdurchsatz in Aufträgen pro Sekunde beim Übertragen dieser Ergebnisse an den 'Hostrechner'.

Auf dem 'Hostrechner' sollten neben dem 'Hostrechner'-Prozeß nur die zur Ausführung dieses Prozesses notwendigen Betriebssystemprozesse gestartet werden. Durch Starten weiterer Prozesse auf dem 'Hostrechner', verringert sich der Durchsatz an der Schnittstelle zum 'Transputerboard'. Daran ändert auch die Vergabe von Prioritäten nichts, da OS/2 das Starten weiterer Anwendungen auch dann erlaubt, wenn bereits Prozesse auf höchster Priorität ablaufen.

7.1.4 Der sekundäre Engpaß

Beseitigt man den primären Engpaß[4], zeigt sich, daß der sekundäre Engpaß der Netz-'Server' ist. Eine obere Schranke für die Auslastung des Ethernet-Segmentes über einen längeren Zeitraum (> 10 Sekunden), bei der der Netz-'Server' alle Dateneinheiten ohne Verluste empfangen kann, ist 5 Megabit/s, also 50%. Wird diese Schranke überschritten, gibt der Prototyp die Warnung an den Benutzer aus.

Der Grenzdurchsatz, bei dem alle Dateneinheiten vom Netz-'Server' empfangen werden können, ist abhängig von der Länge der Dateneinheiten. Dies ist in Abbildung 7.3 dargestellt. Je kleiner die Dateneinheiten sind, desto mehr können in einer Sekunde verarbeitet werden. Es können 1000[5] Ethernet-'Frames' mit einer Brutto-Länge von 310 Byte pro Sekunde empfangen werden. Bei der maximalen Länge eines Ethernet-'Frames' von 1526 Byte[6] konnten nur noch 275 Dateneinheiten pro Sekunde ohne Verluste empfangen werden.

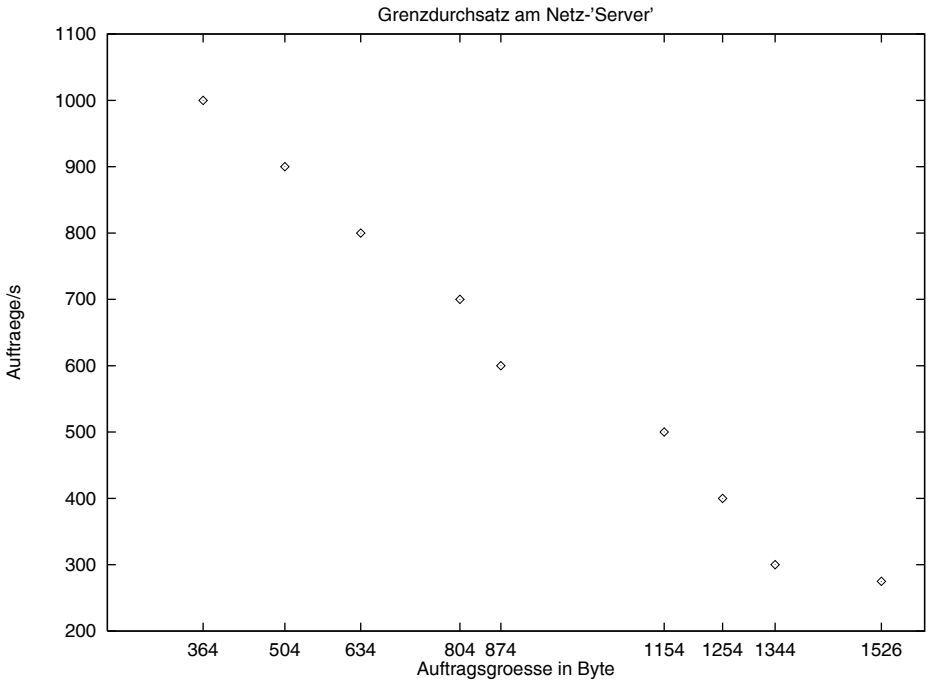


Abbildung 7.3: Grenzdurchsatz in Aufträge/s

[4]Man erreicht dies durch die Angabe 'nur Lastinfo' in dem Dialog aus Abbildung 6.8. Dadurch senden die Paketträger nur noch lokale Statistiken an 'PSTAT'. 'PSTAT' bereitet daraus eine 32 Byte lange Nachricht für den 'Hostrechner'-Prozeß, die einmal pro Sekunde versendet wird. Das liegt natürlich weit unter dem Grenzdurchsatz, so daß hier kein Engpaß auftritt.

[5]Der Lastgenerator konnte mit einer Rate von maximal 1000 Dateneinheiten pro Sekunde betrieben werden.

[6]1500 Byte Nutzdaten zuzüglich Protokoll-'Overhead' (vgl. Abbildung 5.10)

Berechnet man daraus den Durchsatz auf dem Ethernet-Segment, den der Netz-'Server' noch vollständig mitschreiben kann, so ergibt sich folgende Kurve (vgl. Abbildung 7.4).

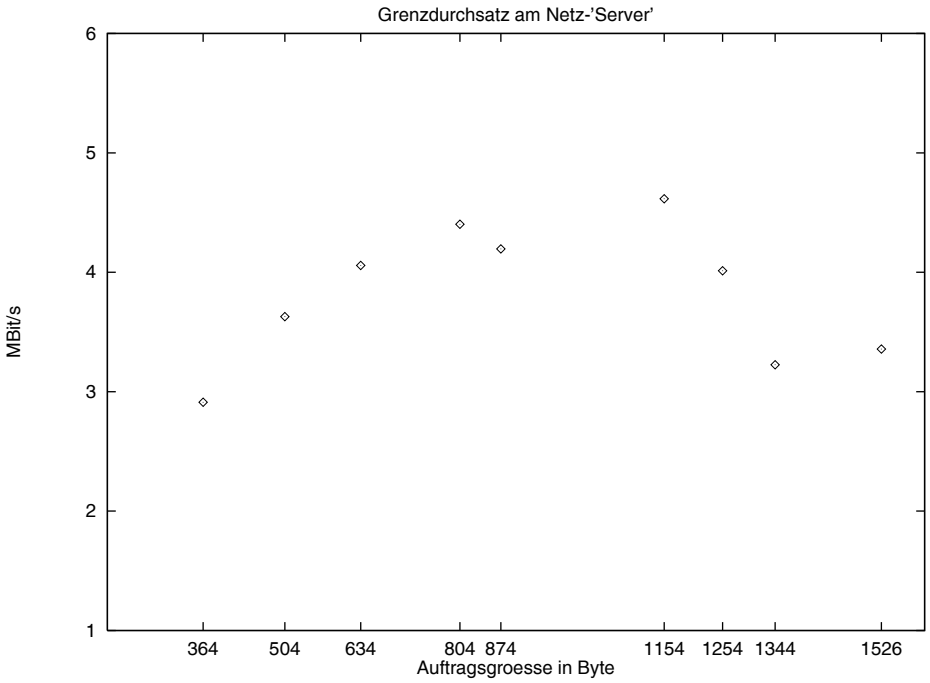


Abbildung 7.4: Grenzdurchsatz in MBit/s

Der Verlauf der Kurve kann nicht definitiv geklärt werden. Vermutlich hängen die Abweichungen von einem linearen Verlauf mit der Pufferverwaltung auf dem Netz-'Server' zusammen. Dort wird der Speicher für eine empfangene Dateneinheit aus kleinen Empfangspuffern zusammengestellt, um den vorhandenen Speicher ökonomischer zu nutzen[7]. Der Verwaltungsaufwand für das Allozieren, Verketten, Belegen und Freigeben dieser Empfangspuffer scheint bei großen Dateneinheiten für das Abfallen der Kurve verantwortlich zu sein.

7.1.5 Maßnahmen während und am Ende der Messung

Dateneinheiten werden grundsätzlich nur von der Ethernet-Schnittstelle vernichtet. Falls nicht mehr ausreichend Platz für eine eintreffende Dateneinheit im Speicher vorhanden ist, vernichtet die Rohdatenerfassung die Dateneinheit und aktualisiert die netznahe Statistik entsprechend. Nach jeder Messung, insbesondere wenn eine der obigen Warnungen auftritt, sollte der Benutzer mit dem Kommando 'status net;' die

[7] Man will nicht für jeden 'Frame' 1518 Byte 'a priori' reservieren.

netznahen Statistiken anfordern. Nur so kann mit Sicherheit festgestellt werden, ob Dateneinheiten während der Messung verloren gegangen sind.

Während der Messung sollte der Benutzer die Auslastung des Ethernet-Segmentes mit Hilfe der Fensters 'MBit/s' beobachten. Tritt eine der obigen Warnungen während eines kurzzeitigen 'Burst' auf, sollte das Meßexperiment nicht vorzeitig abgebrochen werden. Kommt es vor dem primären Engpaß durch stoßartige Last zu einem Stau, wird das durch die Nachrichtenpuffer auf den Paketträgern ausgeglichen. Bei dem sekundären Engpaß bedeutet auch nicht jeder 'Burst' zwingend Datenverlust.

7.2 Bewertung weiterer Komponenten

Ein Teil der Leistungsbewertung wurde bereits im letzten Abschnitt behandelt. Es wurden der primäre und sekundäre Engpaß identifiziert und Anregungen gegeben, was ein Benutzer tun kann, falls ein Engpaß eine Messung gefährdet.

Zwei weitere Komponenten müssen bewertet werden. Einerseits ist der Grenzdurchsatz beim Filtern zu bestimmen. Andererseits muß die Leistungsfähigkeit eines Paketträgers bewertet werden. Unmittelbar einsichtig ist, daß beide Komponenten zum Engpaß werden können. Dazu ist es beim Filtern nur nötig, eine sehr große Anzahl an Vergleichen durchzuführen. Ähnlich verhält es sich mit den Paketträgern. Durch Erweitern der Funktionalität kann man so viel Komplexität in den Analysevorgang hineinbringen, daß ein einzelner Paketträger zum Engpaß des Systems werden kann. In den beiden folgenden Abschnitten wird deshalb der Grenzdurchsatz an diesen Komponenten für verschiedene Aufträge angegeben, um die Leistungsfähigkeit nach oben abzuschätzen.

7.2.1 Grenzdurchsatz beim Filtern

Der kürzeste Auftrag beim Filtern ist ein einzelner Vergleich mit einer Länge von einem Byte. Realistischer sind jedoch mehrere und auch längere Vergleiche. Viele Vergleiche, die sich auf Protokolle der TCP/IP-Familie beziehen, sind beispielsweise zwei Byte lang. In Abbildung 7.5 ist daher der Grenzduchsatz für verschiedene Filter-Aufträge angegeben. Die Kurven sind mit "fxy.data" beschriftet. Es handelt sich dabei um Filter mit x Vergleichen über eine Länge von jeweils y Byte.

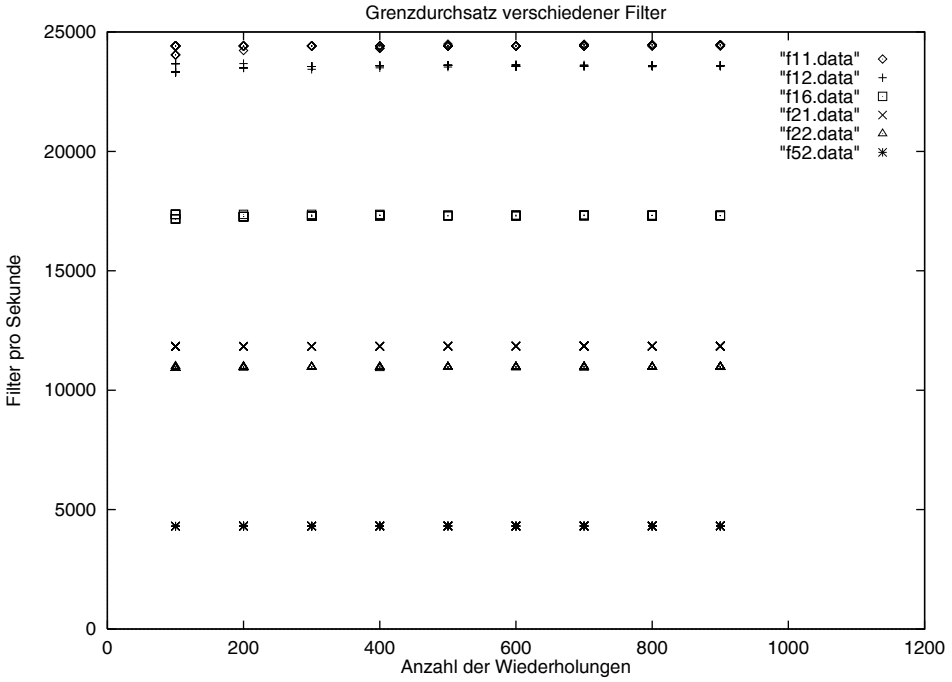


Abbildung 7.5: Grenzdurchsatz bei verschiedenen Filter-Aufträgen

Erwartungsgemäß sinkt der Durchsatz mit höherer Filterkomplexität. Die Filterkomplexität wird primär von der Anzahl der Vergleiche bestimmt (vgl. f11, f21, f52). Innerhalb der gleichen Anzahl an Vergleichen kann man dann den Einfluß der Länge eines Vergleichs auf den Durchsatz erkennen. Der Grenzdurchsatz beim Filtern liegt bei 24468 Filter-Aufträge pro Sekunde. Die in der Kurve "f11.data" deutlich zu erkennenden Unterschiede in den Meßwerten für 100 und 200 Wiederholungen ergeben sich aus der Meßgenauigkeit. Es konnte auf $64\mu s$ genau gemessen werden[8].

Bei 10 MBit pro Sekunde können maximal 17361 'Frames' minimaler Länge innerhalb einer Sekunde über das Ethernet-Segment versendet werden. Dieses theoretische Maximum wird aufgrund von Kollisionen in der Praxis nie erreicht werden. Dennoch könnten alle 'Frames' gefiltert werden, wenn man nur einen Vergleich pro 'Frame' durchführt. Bei komplexeren Vergleichen könnten die Filter u.U. zu einem (dritten) Engpaß werden. Das ist jedoch sehr unwahrscheinlich, da nicht nur 'Frames' minimaler Länge versendet werden und mit zunehmender Last auch die Wahrscheinlichkeit für Kollisionen steigt. Dadurch wird der Grenzdurchsatz eines Ethernet-Segmentes von 17361 'Frames' pro Sekunde in der Praxis weit unterschritten.

Sollten die Filterfunktionen jedoch zu einem Engpaß werden, gibt es zwei einfache Möglichkeiten, diesen Engpaß zu beseitigen. Einerseits bietet es sich an, den 'P-Code'

[8]Dies entspricht der Auflösung des niedrigpriorisierten Zeitgebers. Durch einen hoch priorisierten Zeitgeber mit einer Auflösung von einer Mikrosekunde wäre zu viel 'Overhead' in die Messungen eingegangen.

der Filterfunktionen (vgl. Abschnitt 5.3.5.1) in Maschinencode für die Transputer zu übersetzen. Dies würde sich besonders gut auf den Grenzdurchsatz der komplexeren Filter mit vielen Vergleichen auswirken, da der 'P-Code' hierfür am aufwendigsten ist. Die geschachtelten Funktionsaufrufe beim Interpretieren des 'P-Codes' sind deutlich teurer als der Vergleich der Bitmuster. Durch Maschinencode wäre ein fünffacher Grenzdurchsatz bei komplexen Filtern wahrscheinlich.

Andererseits könnte man die Filterfunktionen in die Paketträger einbinden. Bei mehr als einem Paketträger wäre dadurch eine Leistungssteigerung wahrscheinlich. Der Grenzdurchsatz der Paketträger (s.u.) würde sich jedoch durch den zusätzlichen Filteraufwand verringern.

7.2.2 Grenzdurchsatz des Paketträgers

Zum Messen der Analysezeiten für verschiedene Dateneinheiten wurde durch Filter jeweils ein bestimmter Auftragstyp aus Tabelle 7.1 ausgewählt. Durch eine Nachricht an den Paketträger konnte dann die Analysezeit für eine bestimmte Anzahl von Analysen *derselben* Dateneinheit gemessen werden.

Der kürzeste Auftrag bei der Analyse ist das Berechnen der Auslastung des Ethernet-Segmentes. Die dafür benötigte Zeit ist vernachlässigbar klein. Werden die Dateneinheiten hingegen vollständig[9] ausgewertet, so ergeben sich signifikante Unterschiede in den Analysezeiten der Protokolle bzw. Protokollkombinationen aus Tabelle 7.1. Der Grenzdurchsatz ist in Abbildung 7.6 angegeben. Die Beschriftung der Kurven hat die Form 'px.data'. 'x' bezeichnet die analysierten Protokolle. Steht hier ein 'e', wurden Analysedaten für Ethernet erzeugt. Ein 'i' steht analog für 'IP' und ein 'u' für 'UDP'. Es ergeben sich die gleichen Kombinationsmöglichkeiten wie in Tabelle 7.1.

Die benötigten Zeiten für die Abarbeitung von 100–900 Aufträge wichen bei den verschiedenen Meßwiederholungen maximal $3\mu s$ voneinander ab. Die Aufträge wurden von einem hochpriorisierten 'Thread' abgearbeitet, so daß die benötigte Zeit bis auf eine Mikrosekunde genau gemessen werden konnte. Die minimalen Abweichungen bei den Messungen ergeben sich aus der unterschiedlichen Auslastung der DMA-Schnittstelle des Transputerlinks, über den Dateneinheiten von 'PDEV' empfangen wurden, während die Messungen durchgeführt wurden.

[9] Vollständig bedeutet, daß der Prototyp alle bisher realisierten Analysefunktionen für die Protokolle aufruft.

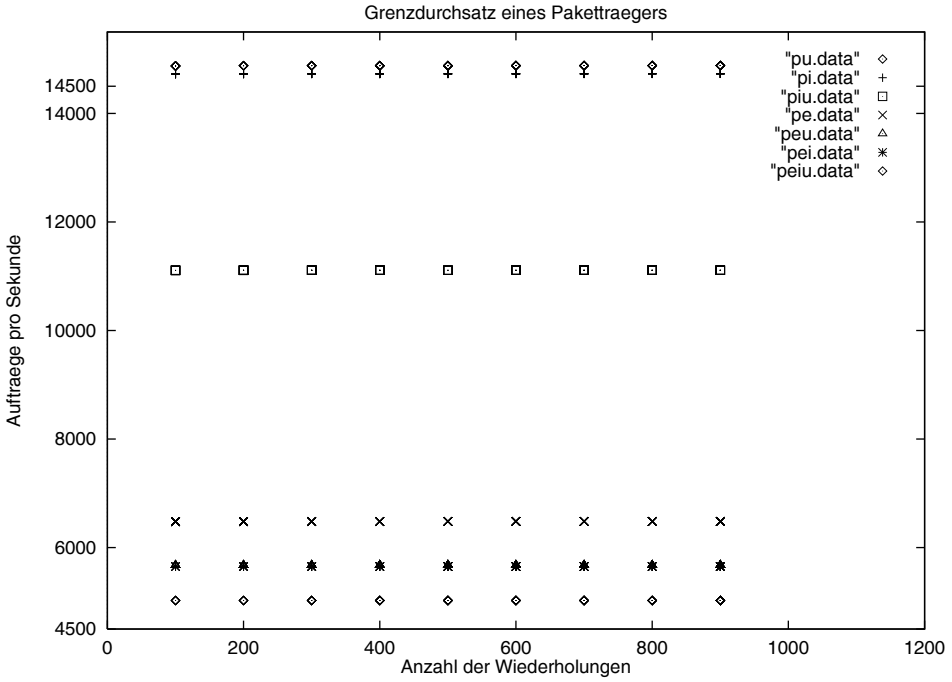


Abbildung 7.6: Grenzdurchsatz eines Paketträgers

Der Grenzdurchsatz eines Paketträgers liegt bei 14880 UDP-Dateneinheiten pro Sekunde ('pu.data'). Wird hingegen der Protokollstapel so weit wie möglich ausgewertet[10], sinkt der Grenzdurchsatz auf 5024 Dateneinheiten pro Sekunde ('peiu.data').

Offensichtlich ist die in Abschnitt 4.4.2.2 gemachte Unterscheidung zwischen *Erkennen* und *Analysieren* auch von praktischer Relevanz. Das Erkennen eines Protokolls unabhängig von der Analyse vorzunehmen, ermöglicht den hohen Durchsatz bei der IP- und UDP-Analyse. Die UDP-Analyse ist schneller als die IP- und Ethernet-Analyse, obwohl der Erkennungsaufwand bei UDP am größten ist. Die deutlichen Unterschiede im Grenzdurchsatz zwischen IP- und UDP-Analyse einerseits und Ethernet-Analyse andererseits lassen sich nur anhand der Programmquellen für die Analysefunktionen erklären. Vermutlich wird die ausschließlich bei der Ethernet-Analysefunktion verwendete 'switch'-Kontrollanweisung (vgl. [KR92, S. 220]) vom Übersetzer sehr schlecht übersetzt. Die Analysefunktionen für Ethernet-Dateneinheiten sollte optimiert werden.

Aussagekräftiger als dieser Grenzdurchsatz ist jedoch die Beobachtung, daß bei allen bisher durchgeführten Messungen, insbesondere bei denen, die zur Identifizierung des primären und sekundären Engpaß führten, alle Dateneinheiten von einem Paketträger rechtzeitig ausgewertet werden konnten[11].

[10] Zur Zeit bedeutet das, Analysedaten für Ethernet, IP und UDP zu generieren.

[11] Durch Ändern der Vermittlungsinformationen auf 'PDEV' kann man alle Dateneinheiten zu einem Paketträger leiten.

Der Lastgenerator, der bei diesen Messungen eingesetzt wurde, konnte maximal 1000 Dateneinheiten pro Sekunde generieren. Das entspricht nicht einmal 20% des Grenzdurchsatzes bei den rechenintensivsten Aufträgen an einem Paketträger (s.o.). Betrachtet man die beiden Paketträger der Konfiguration als eine 'Analyse'-Bedienstation mit zwei Bedieneinheiten, so kann man feststellen, daß die (mittlere) Auslastung dieser Funktionseinheit bei den Messungen zum primären und sekundären Engpaß immer kleiner als 10% war.

7.3 Leistungsprognose

Typischerweise wird die Durchsatzsteigerung durch Parallelverarbeitung mit Hilfe des 'Speedup' angegeben. Der Gewinn (G) durch die Parallelverarbeitung wird auf Grundlage des schnellsten sequentiellen Algorithmus für ein gegebenes Problem berechnet. Dazu wird die Zeit für die sequentielle Berechnung gemessen (T_s) und mit der Zeit für die parallele Berechnung (T_p) auf n Prozessoren verglichen:

$$G(n) = \frac{T_s}{T_p(n)}, n > 0 \tag{7.1}$$

Idealerweise entspricht der Gewinn der Anzahl der eingesetzten Prozessoren. Dieser 'lineare Speedup' wird in der Praxis selten erreicht, da mit jedem weiteren Prozessor der 'Kommunikationsoverhead' zunimmt.

Die Bewertung der Paketparallelität mit einer 'Gewinnkurve' bringt kaum neue Erkenntnisse. Unter der Voraussetzung, daß der sekundäre Engpaß beseitigt wird und genügend Rohdaten für die Analyse zur Verfügung stehen, bekommt man bei der Paketparallelität einen linearen Gewinn[12].

Interessanter ist die Frage, wieviele Paketträger[13] in Abhängigkeit vom Analyseaufwand benötigt werden, um bei maximaler Auslastung des Netzes alle Dateneinheiten auszuwerten.

7.3.1 Anzahl der Prozessoren für Ethernet-Analyse

Die Brutto-Übertragungsgeschwindigkeit von Ethernet beträgt 10 Megabit pro Sekunde. Die minimale Länge einer Ethernet-Dateneinheit ('Frame') beträgt 72 Byte (inklusive der Präambel, vgl. Abbildung 5.10). Aus $10^7 \frac{Bit}{Sekunde} / (72\text{Byte} * 8 \frac{Bit}{Byte})$ ergibt sich ein Grenzdurchsatz von 17361 'Frames' pro Sekunde.

Aufgrund des Grenzdurchsatzes eines Paketträgers, in Abhängigkeit von den Aufträgen aus Tabelle 7.1, ergibt sich der Bedarf an Prozessoren aus Abbildung 7.7.

[12] Man betrachtet nur die Prozessoren mit 'Paketträger'-Prozessen.

[13] Im folgenden wird (Paketträger-) Prozeß mit Prozessor gleichgesetzt. Es wird angenommen, daß für jeden weiteren Paketträger ein weiterer Prozessor zur Verfügung steht.

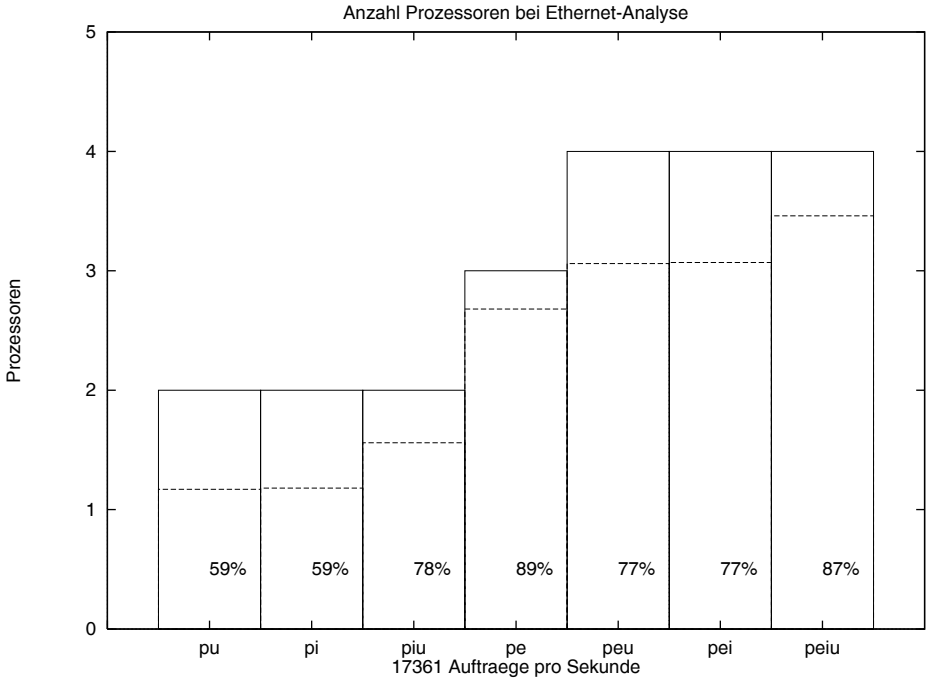


Abbildung 7.7: Anzahl Prozessoren für Ethernet-Analyse

Die Prozentangabe gibt die tatsächliche Auslastung der Prozessoren in Abhängigkeit von der Last (17361 Aufträge/s) und der Auftragsklasse an. Mit vier Prozessoren wären alle Dateneinheiten rechtzeitig analysierbar. Die Auslastung von 87% bietet Raum für weitere (aufwendigere) Analysen.

7.3.2 Anzahl der Prozessoren für FDDI-Analyse

Das 'Fiber Distributed Data Interface' (FDDI) ist ein HSN mit einer Übertragungsgeschwindigkeit von 100 Megabit pro Sekunde. In diesem Abschnitt wird eine Leistungsprognose für die FDDI-Analyse auf Grundlage der Meßwerte in Abschnitt 7.2.2 erstellt. Für die Leistungsprognose sollen folgende Annahmen gelten:

- Der Prototyp wird mit T9000 Transputern realisiert. Ein T9000 hat eine Spitzenleistung von 200 MIPS und 100 MBit/s Übertragungsgeschwindigkeit pro Link. Daraus ergibt sich eine Leistungssteigerung um den Faktor 6,6 bei Berechnungen und um den Faktor 5 bei der Interprozessorkommunikation[14].
- Eine FDDI-'Medium Access Control'(MAC) Dateneinheit zur Übertragung von Benutzerdaten ist 28 Byte lang[15]. Da diese Dateneinheit über keine Informationen verfügt, die eine eingebettete Dateneinheit identifizieren, wird angenommen,

[14] 30 MIPS pro T800 und 200 MIPS pro T9000 sind Herstellerangaben.
[15] vgl. [BHK94, S. 343]

daß grundsätzlich fünf Byte für ein 'Sub-Network Access Protocol'(SNAP)[16] in eine FDDI-Dateneinheit eingebettet werden. Die Informationen dieser fünf Byte werden zur Identifizierung einer IP-Dateneinheit (20 Byte) benötigt, in die wiederum eine UDP-Dateneinheit minimaler Länge (8 Byte) eingebettet wird. Für die weiteren Berechnungen wird demnach eine FDDI-Dateneinheit mit einer Länge von 61 Byte angenommen.

- Der Analyseaufwand für die FDDI-MAC-Dateneinheit und der eingebetteten SNAP-Informationen entspricht der Analyse einer Ethernet-Dateneinheit[17]. Ein Paketträgerprozeß auf einem T9000 hat einen um den Faktor 6,6 größeren Grenzdurchsatz bei den Aufträgen aus Tabelle 7.1.

Es können maximal $10^8 \frac{Bit}{Sekunde} / (61\text{Byte} * 8 \frac{Bit}{Byte}) = 204918$ 'Frames' pro Sekunde übertragen werden. Daraus ergibt sich folgender Bedarf an Prozessoren für die FDDI-Analyse:

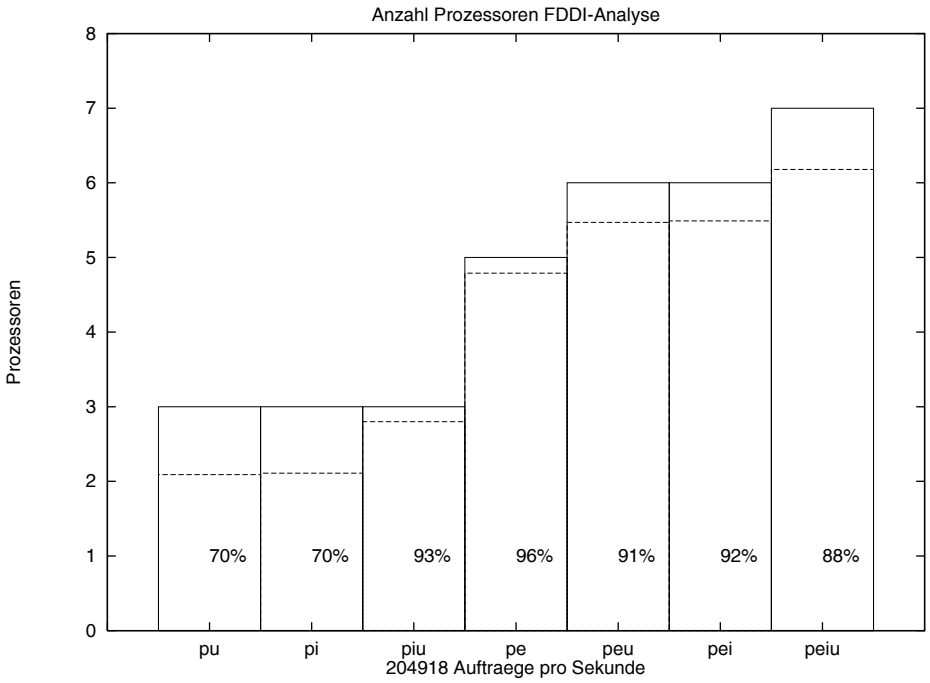


Abbildung 7.8: Anzahl Prozessoren für FDDI-Analyse

Trotz schnellerer Hardware werden mehr Prozessoren (bis zu sieben) für die Analyse von FDDI-Dateneinheiten benötigt. Außerdem werden mindestens drei Prozessoren benötigt und der Raum für Erweiterungen der Analysefunktionalität ist im Mittel kleiner als bei der Ethernet-Analyse (vgl. Abbildung 7.7).

[16] vgl. [BHK94, S. 100]

[17] Es wird angenommen, daß der Grenzdurchsatz dem aus Abbildung 7.6 für 'pe.data' entspricht.

Kapitel 8

Zusammenfassung und Ausblick

In diesem Kapitel werden die wichtigsten qualitativen Ergebnisse der Diplomarbeit zusammengefaßt (vgl. Abschnitt 8.1). In Abschnitt 8.2 wird auf die Erfahrungen bei der Realisierung des Prototypen eingegangen. In einem Ausblick (vgl. Abschnitt 8.3) werden zwei offene Fragen angesprochen, die bei zukünftigen Werkzeugen der Kommunikationsmeßtechnik beachtet werden müssen.

8.1 Zusammenfassung der Ergebnisse

Die Themen der einzelnen Kapitel werden rückblickend in Abschnitt 8.1.1 zusammengefaßt. Danach folgt eine Darstellung der wichtigsten Ergebnisse dieser Diplomarbeit.

8.1.1 Ein Rückblick auf die Diplomarbeit

Die Kommunikationsmeßtechnik wurde als Analysemethode für Kommunikationssysteme vorgestellt. Es folgte eine Darstellung der verschiedenen Aufgaben von Meßwerkzeugen und eine Diskussion typischer Klassifikationen dieser Werkzeuge (vgl. Kapitel 1). In Kapitel 2 folgte die Diskussion der Anforderungen an Meßwerkzeuge. Die wichtigsten Anforderungen ergaben sich aus der Meßaufgabe.

Im Kontext von Hochgeschwindigkeitsnetzen wurden weitere Anforderungen diskutiert. Es wurde herausgearbeitet, daß durch den enormen Zuwachs an Übertragungsgeschwindigkeit zukünftig besonders hohe Anforderungen an die Erweiterbarkeit, Skalierbarkeit und Leistungsfähigkeit der Meßwerkzeuge gestellt werden. In diesem Zusammenhang wurde die Parallelverarbeitung als eine Möglichkeit vorgestellt, Meßwerkzeuge an die hohen Übertragungsgeschwindigkeiten zukünftiger Netze anzupassen. Das zweite Kapitel wurde mit einer Aufstellung der Anforderungen an ein paralleles Meßwerkzeug für Hochgeschwindigkeitsnetze abgeschlossen.

In Kapitel 3 wurden Konzepte der Parallelverarbeitung diskutiert. Neben einer Diskussion der Zuordnungsproblematik, die ein zentraler Aspekt jeder Parallelverarbeitung ist, wurde speziell auf die parallele Protokollverarbeitung eingegangen. Die Motivation für die Diskussion verschiedener Ansätze der parallelen Protokollverarbei-

tung war die Idee, eines dieser Konzepte für die parallele Protokollanalyse anzupassen. Ein weiterer Schwerpunkt des dritten Kapitels bildete die Darstellung der verfügbaren Parallelrechner und der Restriktionen durch diese Hardware.

Der Entwurf eines parallelen Meßwerkzeuges wurde im vierten Kapitel dargestellt. Es zeigte sich, daß der Entwurf von den Restriktionen durch die Hardware einerseits und dem gewählten Konzept zur Parallelverarbeitung andererseits geprägt ist. Während des Entwurfs wurden zwei verschiedene Klassen von Funktionen vorgestellt. Die essentiellen Funktionen wurden entworfen, um die wesentlichen parallel auszuführenden Aufgaben des Meßwerkzeuges zu realisieren. Die unterstützenden Funktionen wurden entworfen, um die 'Infrastruktur' zu realisieren, auf der die essentiellen Funktionen aufbauen. Die einzelnen Entwurfsentscheidungen wurden mit den Anforderungen an den Prototypen und den Restriktionen durch die Hardware begründet. Es konnte eine erste Grobgliederung des Prototypen durch Prozesse vorgestellt werden.

In Kapitel 5 wurden die Prozesse auf Grundlage eines allgemeinen 'Prozeß-Template' verfeinert. Das Zusammenwirken der einzelnen Funktionen in den Prozessen wurde anhand der Implementation diskutiert. Die wichtigsten Algorithmen wurden mit Nassi-Schneiderman Diagrammen dargestellt und im Detail diskutiert. Es wurde eine Methode für systematische Erweiterungen — zukünftige Realisierungen — vorgeschlagen und es wurden die wichtigsten Erweiterungen[1] benannt. Mit der Darstellung der Unterschiede zwischen dem Prototypen und einem anderen Werkzeug wurde das fünfte Kapitel abgesschlossen.

Die Darstellung der Benutzerschnittstelle und zwei Fallbeispiele für Messungen mit dem Prototypen waren Gegenstand des sechsten Kapitels. Die Beispiele zeigten das Vorgehen zur Durchführung von Meßexperimenten.

Im siebten Kapitel wurde die Leistung des Prototypen bewertet. Es wurden zwei Engpässe identifiziert, die durch modernere Hardware beseitigt werden können. Anschließend wurde der Grenzdurchsatz der Filter und der parallelen Protokollanalyse mit systematisch generierten Aufträgen gemessen. Diese Meßergebnisse bildeten die Grundlage für eine Leistungsprognose beim Einsatz des Prototypen in einem Hochgeschwindigkeitsnetz.

8.1.2 Die wichtigsten Ergebnisse

Die wichtigsten Ergebnisse können erst an dieser Stelle, nach der Leistungsbewertung in Abschnitt 7, vorgestellt werden. Bei der Leistungsbewertung hat sich gezeigt, daß man trotz veralteter Hardware mit Parallelverarbeitung in der Lage ist, typische Meßaufgaben der Kommunikationsmeßtechnik 'on-line' zu bewältigen.

Der Hauptgrund liegt in der Wahl eines geeigneten, skalierbaren Konzeptes zur parallelen Protokollanalyse. Die Paketparallelität (vgl. Abschnitt 3.3.3.5) hat sich bei den durchgeführten Messungen bewährt. Es zeigte sich, daß die Schnittstelle zum Ethernet vor der parallelen Protokollanalyse zum Engpaß wird. Der Grenzdurchsatz

[1] Zumindest nach Meinung des Autors handelt es sich dabei um die wichtigsten Erweiterungen.

eines Paketträgers wurde bei Messungen in einem typischen Ethernet-Segment nicht einmal annähernd erreicht.

Die Messungen für die Leistungsbewertung haben aber auch gezeigt, daß der Prototyp noch verbessert werden kann. Beispielsweise sollten die Filterfunktionen beschleunigt werden. Außerdem deuten die starken Unterschiede in den Grenzdurchsätzen bei verschiedenen Analyseaufträgen an die Paketträger daraufhin, daß die Analysefunktion für Ethernet-'Frames' verbessert werden kann (vgl. Abbildung 7.6).

Ein weiteres wichtiges Ergebnis ist die Erkenntnis, daß Prozesse und Funktionen, die auf Grundlage von 'Templates' realisiert werden, den Entwicklungsaufwand deutlich reduzieren. Die manuelle Zuordnung von Prozessen zu Prozessoren wird einfacher, wenn man gleichartige Prozesse auf Prozessoren plazieren kann. Die Entwicklung des 'Prozeß-Templates' auf Grundlage einer allgemeinen Nachrichtenvermittlung war dabei der wesentliche Abstraktionsschritt (vgl. Abschnitt 5.2.1).

Die Leistungsprognose im Abschnitt 7.3.2 hat schließlich gezeigt, daß die Parallelverarbeitung bei Meßwerkzeugen für Hochgeschwindigkeitsnetze nicht nur hilfreich sondern auch notwendig ist. Aufgrund des Netz-Zugriffsverfahrens bei FDDI kommt es auch unter hoher Last zu keinen Kollisionen (wie etwa bei Ethernet). Die maximale Auslastung eines FDDI-Netzes kann daher das theoretische Maximum — auch in der Praxis — erreichen. Da für die Leistungsprognose die Rechenleistung eines modernen Prozessors vorausgesetzt wurde, wird deutlich, daß Einprozessor-Systeme zukünftig nicht in der Lage sein werden, vergleichbare Messungen in einem HSN durchzuführen.

Die Anforderungen aus Abschnitt 2.7.4 konnten erfüllt werden. Während die Anforderungen bezüglich der 'Standardauswertung' und der graphischen Benutzeroberfläche primär der Präsentation des Werkzeuges dienten, sicherten die Anforderungen zur Analyse von Protokollen der TCP/IP-Familie, daß der Prototyp praxisrelevante Rohdaten analysieren kann.

8.2 Erfahrungen bei der Realisierung

Die Erfahrungen bei der Realisierung des Prototypen lassen sich in drei Kategorien einteilen:

- Probleme mit der Hardware
- Probleme bei der Programmierung
- Gruppenarbeit

8.2.1 Probleme mit der Hardware

Die Restriktionen durch die Hardware wurden in Kapitel 3 beschrieben. Daneben gab es eine Reihe von Problemen, die in der Anfangsphase der Diplomarbeit gelöst werden mußten. Beispielsweise mußte für die Koppelung der verwendeten 'Transputerboards'

ein passendes Kabel gebastelt werden, da die TBs von verschiedenen Herstellern stammen. Die potentielle Vernichtung der teuren Hardware vor Augen, ließ den Moment des ersten Einschaltens der Konfiguration zu einem unvergeßlichen Erlebnis werden.

Außerdem mußte ein Programm zur Programmierung des Kreuzschienenverteilers des TB entwickelt werden. Ein vergleichbares Programm wurde leider nicht mit dem TB ausgeliefert. Ein vergleichbares Problem trat bei dem TB mit der Ethernet-Schnittstelle auf. Es wurden zwar Programme zur Ansteuerung der Schnittstelle mitgeliefert, jedoch nicht für den von uns verwendeten Compiler. Der 'Device Driver' für die Ethernet-Schnittstelle mußte an unsere Entwicklungsumgebung angepaßt werden.

Die Anfangsphase der Diplomarbeit wurde dazu verwendet, eine Infrastruktur für die Programmierung der Parallelrechner zu schaffen, auf der weitere Arbeiten im Arbeitsbereich Rechnerorganisation aufbauen können.

Ein weiteres Problem ergab sich aus einem veralteten 'Hostrechner', der in der Anfangsphase eingesetzt werden mußte. Viel Zeit wurde darauf verwendet, die geringe Leistungsfähigkeit dieses Rechners auszugleichen. Glücklicherweise wurde dann für die Diplomarbeit ein leistungsfähiger 'Hostrechner' zur Verfügung gestellt, der es erlaubte, OS/2 als Betriebssystem einzusetzen. Dadurch ergaben sich neue Möglichkeiten, beispielsweise die Fenster-basierte Benutzerschnittstelle des Prototypen.

8.2.2 Probleme bei der Programmierung

Probleme bei der Programmierung ergaben sich aus mangelhafter oder veralteter Dokumentation der Entwicklungsumgebung und des Betriebssystems. Viel Zeit mußte darauf verwendet werden, die programmierten Funktionen zu überprüfen. Ein ständiges Mißtrauen[2] gegenüber dem 'Parallel-C'-Compiler begleitete die Implementierung. Ein Zitat aus der Dokumentation mag für sich selbst sprechen: *"[...] this release is closer to Ansi-C..."*.

Außerdem war kein 'Debugger'[3] für den 'Parallel-C'-Compiler vorhanden. Es mußte besonders viel Zeit investiert werden, um jede Funktion mit eigenen 'Debug-Informationen' auszustatten, die zumindest eine rudimentäre Einsicht in den Ablauf der Prozesse ermöglichten. Da diese Informationen an den 'Hostrechner' gesendet werden mußten, war die Implementierung der Nachrichtenvermittlung besonders schwer. Entweder funktionierte das Senden und Empfangen von Nachrichten oder es gab 'irgendwo' einen oder mehrere Fehler.

Bei der Programmierung mußte darüber hinaus zwischen 'Systemprogrammierung' der Transputer und 'Anwendungsprogrammierung' unter OS/2 gewechselt werden. Die Systemprogrammierung der Transputer machte den besonderen Reiz dieser Diplomarbeit aus. Die Anwendungsprogrammierung unter OS/2 war eher notwendig als reizvoll. Die Forderung nach einer graphischen Benutzerschnittstelle ist heutzutage üblich. Der damit verbundene Aufwand wird jedoch leicht unterschätzt. Die Pro-

[2] Rückblickend war dieses Mißtrauen übertrieben. Unbegründet war es jedoch nicht. Es wurde ein gravierender Fehler in der Entwicklungsumgebung gefunden.

[3] Ein Werkzeug zur Fehlersuche in Programmen. Es ermöglicht die Einzelschrittausführung eines Programms und das Anzeigen der Veränderungen an Variablen.

grammierung (Einarbeitung und Implementation) der Benutzerschnittstelle war zeitaufwendig und teuer[4]. Die Programmierung der rudimentären Analysefunktionen für die Transputer dauerte drei Tage. Die Programmierung der rudimentären Funktionen für die Benutzerschnittstelle dauerte drei Wochen.

Besonders hilfreich bei der Bewältigung der vielen Probleme war der Kontakt zu einer Vielzahl hochqualifizierter Personen über Interessengruppen in den 'Net News'[5]. Durch die vielen Interessengruppen lassen sich vermeintlich unlösbare Probleme darauf reduzieren, die richtigen Fragen zu stellen und Informationen zu recherchieren.

8.2.3 Gruppenarbeit

Bei der Programmierung der Transputer erwies sich die Gruppenarbeit mit einem anderen Studenten[6] als besonders vorteilhaft. Ein regelmäßiger Erfahrungsaustausch bewahrte uns vor einer vielzahl unnötiger Fehler und Fehlentscheidungen bei der Programmierung.

8.3 Offene Fragen

Mögliche Erweiterungen des Prototypen wurden bereits am Ende des fünften Kapitels diskutiert. An dieser Stelle soll deshalb nur noch auf zwei offene Fragen eingegangen werden.

8.3.1 Die Schnittstelle für Messungen in Hochgeschwindigkeitsnetzen

Es gibt viele neue Zugriffsverfahren im Bereich der Hochgeschwindigkeitsnetze. Einige dieser Zugriffsverfahren erschweren die Rohdatenerfassung durch Meßwerkzeuge erheblich. Ein in diesem Zusammenhang unproblematisches HSN ist FDDI (vgl. Abschnitt 8.3.1.1). Der problematische Fall kann am Beispiel von ATM diskutiert werden.

8.3.1.1 Meßdatenerfassung in einem FDDI-Netz

Die Stationen werden in einem FDDI-Netz ringförmig angeordnet (vgl. [BHK94, S. 343ff]). Für die Meßdatenerfassung entscheidend ist, daß alle Dateneinheiten den physikalischen Ring vollständig umrunden, bevor die sendende Station ihre eigenen Dateneinheiten wieder vom Netz entfernt.

Ein Meßwerkzeug kann daher unabhängig von der Position im Ring den gesamten Verkehr beobachten. Es können insbesondere Verkehrsmatrixen erstellt werden.

[4] Es mußte Literatur angeschafft werden.
[5] Ein weltweiter Informations-Austauschdienst.
[6] Namentlich Herr Gerrit Möller.

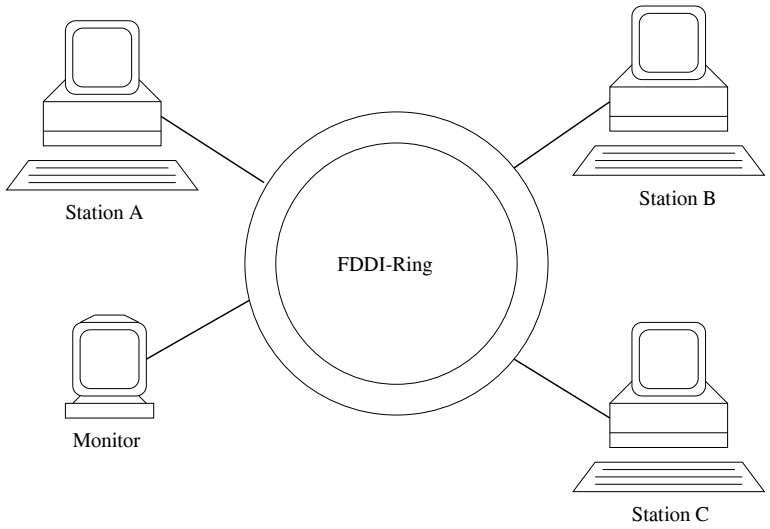


Abbildung 8.1: FDDI-LAN

8.3.1.2 Meßdatenerfassung in einem ATM-Netz

'Asynchronous Transfer Mode' (ATM) bedeutet, daß ununterbrochen Dateneinheiten unabhängig von Benutzerdaten in einem ATM-Netz versendet werden. Falls keine Benutzerdaten vorhanden sind, generiert die ATM-Schnittstelle 'leere' Dateneinheiten, die versendet werden (vgl. [BHK94, S. 427]).

In Abbildung 8.2 ist ein typisches ATM-LAN abgebildet. Die Stationen sind über einen Vermittlungsknoten sternförmig angeordnet. Durch diese Topologie hat die Positionierung des Meßwerkzeuges entscheidenden Einfluß auf den beobachtbaren Verkehr im Netz. Bei einer Ankoppelung an Position (A) kann beispielsweise nur der Verkehr von und zu Station (1) beobachtet werden. Das gesamte Verkehrsaufkommen im LAN kann nur beobachtet werden, wenn das Werkzeug in den ATM-Vermittlungsknoten integriert wird (Position (B)).

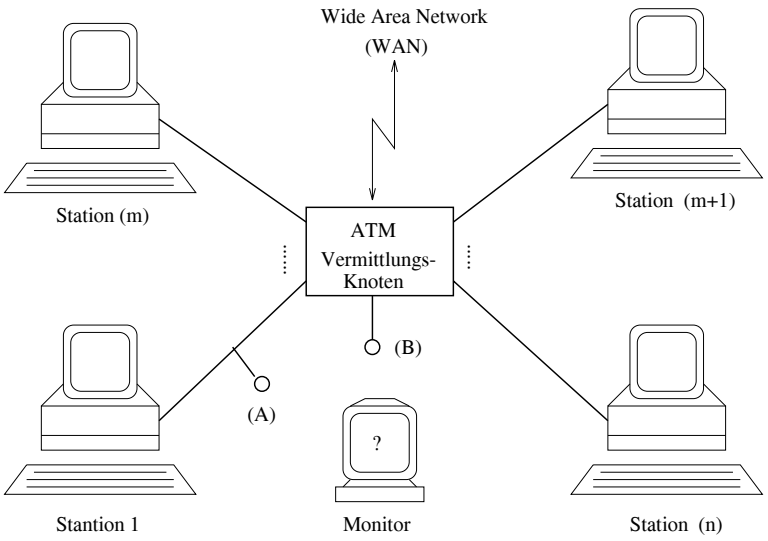


Abbildung 8.2: Typisches ATM-LAN

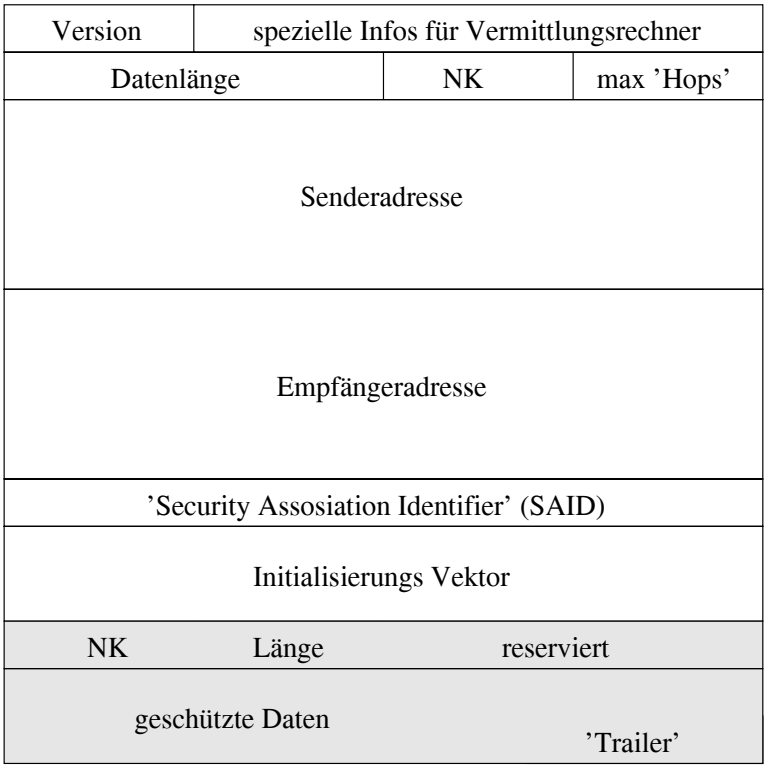
Ein ähnliches Problem ergibt sich bei Messungen in 100VG-AnyLAN[7] Netzen. Hier kann überhaupt keine globale Sicht auf das Verkehrsaufkommen durch ein Meßgerät erreicht werden. Die Kommunikation in einzelnen 'Ästen' dieser Baumtopologie ist weitgehend unabhängig voneinander. Eine Lösung könnte eine verteilte Datenerfassung sein. An jeden 'Hub' müßte ein Modul zur Rohdatenerfassung gekoppelt werden.

8.3.2 Neue Protokolle mit Verschlüsselungsmechanismen

Ein weiteres Problem könnten neue Protokolle darstellen, die die Benutzerdaten teilweise oder vollständig verschlüsseln. Ein Beispiel für ein Protokoll mit Verschlüsselungsmechanismen ist die IP Version 6 (vgl. RFC 1752). Die in Abbildung 8.3 gestrichelten Bereiche der Dateneinheit sind für ein Meßwerkzeug nicht zugänglich.

Glücklicherweise können Daten mit diesem Protokoll auch unverschlüsselt übertragen werden. In der Konfigurations- und Testphase eines Netzes kann ein Meßwerkzeug weiterhin beim Aufspüren von Fehlern helfen. Ist die Verschlüsselung im normalen Netzbetrieb erlaubt, sind auftretende Fehler eventuell nicht mehr maschinell zu erkennen.

[7]Bei dieser Alternative zum 100 Megabit-Ethernet wird ein LAN hierarchisch aufgebaut. Stationen werden über 'Hubs' sternförmig angeordnet. Mehrere 'Hubs' werden hierarchisch miteinander verbunden (vgl. [Epp95]).



NK - nächster Kopf (identifiziert die eingebetteten Daten)

mögliche NK: (IPv4 - Protokoll) und zusätzlich
'Hop-by-Hop Informationen'
'Vermittlungs-Informationen'
'Fragment-Informationen'
'Authentifizierungs-Informationen'
'Geschützte Informationen'
'Ende-zu-Ende-Informationen'

Abbildung 8.3: IPv6 Dateneinheit

Literaturverzeichnis

[ASU88] A.V. Aho, R. Seti, and J.D. Ullman. *Compiler-Bau, Band 1.* Addison-Wesley (Deutschland) GmbH, 1988.

[Ben94] C. Benecke. Möglichkeiten der Beschleunigung der Kommunikation in Hochgeschwindigkeitsnetzen durch Parallelisierung. Studienarbeit, Universität Hamburg, Arbeitsbereich Rechnerorganisation, 1994.

[BHK94] A. Badach, E. Hoffmann, and O. Knauer. *High Speed Internetworking.* Addison-Wesley, 1994.

[Car88] A. Carling. *Parallel Processing. The Transputer and OCCAM.* SIGMA Press, 1988.

[CDK94] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems. Concepts and Design.* Addison-Wesley, 2 edition, 1994.

[Com91] D.E. Comer. *Internetworking with TCP/IP. Principles, Protocols, and Architecture*, volume 1. Prentice-Hall, Englewood Cliffs, New Jersey, 2 edition, 1991.

[CS91] D.E. Comer and D.L. Stevens. *Internetworking with TCP/IP. Design, Implemtation, and Internals*, volume 2. Prentice-Hall, Englewood Cliffs, New Jersey, 2 edition, 1991.

[D+94] P. Dauphin et al. ZM4/Simple: A General Approach to Performance Measurement and Evaluation of Distributed Systems. In T. Casavant and M. Singhal, editors, *Readings in Distributed Computing Systems*, pages 286–309. IEEE Computer Society Press, 1994.

[dM95] H. de Meer. Modelling and Management of responsive systems: the quality of service for multi-media applications in high speed networks as an example. Technical report, Universität Hamburg, Arbeitsbereich RO, 1995.

[Ebe93] H. Ebert. *Transputer und Occam. Das Handbuch für Systementwickler.* Heise, Hannover, 1993.

[Eck93] F. Eckgold. *Systemprogrammierung OS/2.* Vieweg, Braunschweig, Wiesbaden, 1993.

[Epp95] K. Eppele. Spaß ab 100. *IX Multiuser Multitasking Magazin*, (4):80–90, apr 1995.

[F+87] F. Fehlau et al. Messung des Leistungsverhaltens Lokaler Netze mit einem Software Monitor. *Informatik Forschung Entwicklung*, (2):55–64, 1987.

[FSZ83] D. Ferrari, G. Serazzi, and A. Zeigner. *Measurement and Tuning of Computer Systems.* Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[FY94] P.G.S. Florissi and Y. Yemini. Management of Application Quality of Service. Technical Report NY 10027, DCC Lab. Computer Science Dep. Columbia University, New York, 1994.

[GK94] P. Gerdsen and P. Kröger. *Kommunikationssysteme. Theorie, Entwurf, Meßtechnik*, volume 1. Springer-Verlag, 1994.

[GNI93] M.W. Goldberg, G.W. Neufeld, and M.R. Ito. A Parallel Approach to OSI Connection-Oriented Protocols. In B. Pehrson, P. Gunningberg, and S. Pink, editors, *Protocols for High Speed Networks, III*, pages 219–232. North-Holland, 1993.

[Hau88] J.S. Haugdahl. Benchmarking LAN Protocol Analyzers. In *13th Conf. on Local Computer Networks*, pages 375–384, Minneapolis, Minnesota, oct 1988.

[Hei94] H.U. Heiss. *Prozessorzuteilung in Parallelrechnern.* Reihe Informatik, Band 98. BI-Wiss.-Verlag, 1994.

[Hin86] U. Hinrichsen. *Ein Monitor für ein autonomes Datenübertragungssystem.* Informatik-Bericht 86-05, TU Braunschweig, 1986.

[Hol90] A.I. Holub. *Compiler Design in C.* Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[Hol92] P. Holsberg. *UNIX Desktop Guide to Tools.* SAMS (Prentice–Hall), 1992.

[Inm88] Inmos Limited. *Transputer Reference Manual*, 1988.

[Inm90] Inmos Limited. *Transputer Development System*, 2 edition, 1990.

[Int89] International Business Machines Corporation. *IBM Operating System/2, I/O Subsystems and Device Support, Volume 1, Device Drivers*, 1989.

[Jai91] R. Jain. *The Art of Computer Systems Performance Analysis. Techniques for Experimental Design, Measurement, Simulation, and Modeling.* John Wiley & Sons, Inc., 1991.

[Job92] F. Jobst. *Compilerbau: Von der Quelle zum professionellen Assemblertext.* Hanser, München, Wien, 1992.

[JV87] E. Jessen and R. Valk. *Rechensysteme. Grundlagen und Modellbildung.* Springer-Verlag, Berlin, Heidelberg, New York, 1987.

[K+95] R. Klar et al. *Messung und Modellierung paralleler und verteilter Rechensysteme.* B.G. Teubner, Stuttgart, 1995.

[Ker92] H. Kerner, editor. *Rechnernetze nach OSI.* Addison-Wesley, 1992.

[Kie95] M. Kienle. Marke Eigenbau. TIS: Toolkit für anwendungsorientierte Firewall-Systeme. *IX Multiuser Multitasking Magazin*, (6):54–61, jun 1995.

[KR92] B.W. Kernighan and D.M. Ritchie. *Programmieren in C.* Hanser, Prentice-Hall, München, Wien, London, 2 edition, 1992.

[Kus95] M. Kuschke. Teuflisch gut. SATAN: PD-Tool zum Auffinden von Security-Lücken. *IX Multiuser Multitasking Magazin*, (6):54–61, jun 1995.

[Lip88] H. Lipp. Entwurf und Realisierung eines Software-Monitors für lokale Rechnernetze zur Unterstützung von Verkehrs- und Leistungsanalysen. Diplomarbeit, Universität Hamburg, Arbeitsbereich Rechnerorganisation, 1988.

[LMB92] J.R. Levine, T. Mason, and D. Brown. *lex & yacc.* UNIX Programming Tools. O'Reilly & Associates, Inc., 2 edition, 1992.

[MP86] S. McMenamin and J. Palmer. *Strukturierte Systemanalyse.* Hanser, 1986.

[MSS93] M. Mansouri-Samani and M. Sloman. Monitoring Distributed Systems. *IEEE Network*, 7(6):20–30, 1993.

[Mue94] J. Mueller. *The Ultimate OS/2 Programmer's Manual.* McGraw-Hill, 1994.

[MWT94] J.K. Muppala, W. Wang, and K.S. Trivedi. Dependability Evaluation Through Measurement and Models. In R. Marie, G. Haring, and G. Kotsis, editors, *7th Int. Conf. on Modelling Techniques and Tools for Performance Evaluation*, Schriftenreihe der Österr. Computer Gesellschaft, Band 73. 1994.

[Pag91] B. Page. *Diskrete Simulation. Eine Einführung mit Modula-2.* Springer-Verlag, Berlin, Heidelberg, 1991.

[Par95] C. Parris. Using SNMP to Manage Guaranted Performance Connections. Research Report RC 19973 (88451), IBM Research Division, 1995.

[Pen92] V. Penner. *Parallelität und Transputer: Von den Grundlagen zur Anwendung.* Vieweg, Braunschweig, Wiesbaden, 1992.

[Roc90] M.J. Rochkind. *Fortgeschrittene Bildschirmprogrammierung in C unter UNIX und DOS.* Hanser, Prentice-Hall, Wien, London, 1990.

[SF85] A.T. Schreiner and G. Friedman. *Compiler bauen mit Unix. Eine Einführung.* Hanser, München, Wien, 1985.

[SGS91] SGS-THOMSON Microelectronics Group. *IMS B431 Ethernet TRAM*, 1991.

[Sta93] W. Stallings. *Local and Metropolitan Area Networks.* Macmillan Publishing Comp., 4 edition, 1993.

[Tan95] A.S. Tanenbaum. *Verteilte Betriebssysteme.* Prentice Hall, 1995.

[Thu93] E.M. Thurner. Performance Measurement Using System Monitors. In L. Donatiello and R. Nelson, editors, *Performance Evaluation of Computer and Communication Systems*, Lecture Notes in Computer Science 729. Springer-Verlag, 1993.

[Tic85] W. F. Tichy. RCS–A System for Version Control. *Software–Practice & Experience*, 7(15):637–654, jul 1985.

[WF93] C.M. Woodside and G. Franks. Alternative Software Architectures for Parallel Protocol Execution with Synchronous IPC. *IEEE/ACM Transactions on Networking*, 1(2):178–186, apr 1993.

[WK90] B.E. Wolfinger and J.J. Kim. Load Measurements as a Basis for Modeling the Load of Innovative Communication Systems with Service Integration. In *Second IEEE Workshop on Future Trends of Distributed Computing Systems, Cairo, Agypt*, sep/oct 1990.

[Wol93] B.E. Wolfinger. Analyse, Optimierung und Performance-Management von Kommunikationsnetzen und verteilten Anwendungen. In *Proceedings: Networks '93*, 1993.

[You89] E. Yourdon. *Modern Structured Analysis.* Prentice-Hall, 1989.

[Zit91a] M. Zitterbart. *Funktionale Parallelität in transportorientierten Kommunikationsprotokollen.* Fortschritt-Berichte, VDI Reihe 10, Nummer 183. VDI-Verlag, 1991.

[Zit91b] M. Zitterbart. High-Speed Transport Components. *IEEE Network Magazine*, pages 54–63, jan 1991.

[ZK93] C. Zimmermann and A.W. Kraas. *Mach. Konzepte und Programmierung.* Springer-Verlag, Berlin, Heidelberg, New York, 1993.

Abkürzungen

ARP	Address Resolution Protocol, vgl. RFC 826
ASCII	American Standard Code for Information Interchange
ATM	Asynchronous Transfer Mode
BER	Bit Error Rate, Bitfehlerhäufigkeit
CPU	Central Processing Unit
CSMA/CD	carrier sensing, multiple access with collision detection
CUA	Common User Access
DMA	Direct Memory Access
DNS	Domain Name System, vgl. RFC 1034
EA	Empfänger Adresse
FDDI	Fiber Distributed Data Interface
GNU	GNU ist not UNIX
HSN	High Speed Network, Hochgeschwindigkeitsnetz
HW	Hardware
ICMP	Internet Control Message Protocol, vgl. RFC 792
IEEE	Institute of Electrical and Electronics Engineers
IGP	Interior Gateway Protocol, vgl. RFC 1009
IP	Internet Protocol, vgl. RFC 791
ISO	Internationale Standard Organisation
KByte	Kilobyte
LAN	Local Area Network, Lokales Netz
LLC	Logical Link Control
MBit	Megabit
MByte	Megabyte
MEAG	Meßdatenerfassungs- und Analysegerät
MHz	Megaherz
MIB	Management Information Base
MTU	Maximum Transmission Unit
NBO	Network Byte Order
NMS	Network Management System

Abkürzungen (Fortsetzung)

OS/2	Operating System/2
OSI	Open System Interconnection
OSIRM	OSI Reference Model
OSPF	Open Shortest Path First, vgl. RFC 1583
PC	Personal Computer
PD	Public Domain
PDU	Protocol Data Unit, Dateneinheit
PM	Presentation Manager
RAM	Random Access Memory
RARP	Reverse Address Resolution Protocol, vgl. RFC 903
RCS	Revision Control System
RFC	Request for Comments
RMON	Remote Monitoring
RO	Rechnerorganisation
RZ	Rechenzentrum
SA	Absender (Sender Adresse)
SDU	Service Data Unit
SNMP	Simple Network Management Protocol
SW	Software
TB	Transputer Board
TCP	Transmission Control Protocol, vgl. RFC 793
TDR	Time Domain Reflectometer
TELNET	Telnet Protocol, vgl. RFC 854,855
VB	Vermittlungs-bit